From Semantics to Types: the Case of the Imperative λ-Calculus

Ugo de'Liguoro
Department of Computer Science
Università di Torino
Torino, Italy
ugo.deliguoro@unito.it

Riccardo Treglia
Department of Computer Science
Università di Torino
Torino, Italy
riccardo.treglia@unito.it

We propose an intersection type system for an imperative λ-calculus based on a state monad and equipped with algebraic operations to read and write to the store. The system is derived by solving a suitable domain equation in the category of ω-algebraic lattices; the solution consists of a filter-model generalizing the well known construction for ordinary λ-calculus. Then the type system is obtained out of the term interpretations into the filter-model itself. The so obtained type system satisfies the "type-semantics" property, and it is sound and complete by construction.

1 Introduction

Since Strachey and Scott's work in the 60's, λ-calculus and denotational semantics, together with logic and type theory, have been recognized as the mathematical foundations of programming languages. Nonetheless, there are aspects of actual programming languages that have shown to be quite hard to treat, at least with the same elegance as the theory of recursive functions and of algebraic data structures; a prominent case is surely side-effects.

The introduction of notions of computation as monads by Moggi in [22] greatly improved the understanding of "impure", namely non functional features in the semantics of programming languages, by providing a unified framework for treating computational effects: see [26, 27], the introductory [8], and the large body of bibliography thereafter; a gentle introduction and references can be found e.g. in [17].

Starting with [14], we have considered an untyped variant of Moggi's computational λ-calculus, with two sorts of terms: *values* denoting points of some domain D, and *computations* denoting points of TD, where T is some generic monad and $D \cong D \to TD$. The goal was to show how such a calculus can be equipped with an operational semantics and an intersection type system, such that types are invariant under reduction and expansion of computation terms, and convergent computations are characterized by having non-trivial types in the system.

Our approach was limited to a pure calculus without constants, where the unit and bind operations are axiomatized by the monadic laws from [27]. On the other hand, such operations may model how morphisms from values to computations compose, but do not tell anything about how the computational effects are produced. In the theory of *algebraic effects* [23, 24, 25], Plotkin and Power have shown under which conditions effect operators live in the category of algebras of a computational monad, which is isomorphic to the category of models of certain equational specifications, namely varieties in the sense of universal algebra [19].

Toward a general theory of intersection type systems for (untyped) computational λ-calculi with algebraic effects, we have studied in the draft [15] the case of the imperative λ-calculus λ_{imp}, both because the modeling of a functional calculus with (implicitly higher-order) side-effects is of interest per

Ana Sokolova (Eds.): MFPS 2021
EPTCS 351, 2021, pp. 168–183, doi:10.4204/EPTCS.351.11

© U. de'Liguoro & R. Treglia
This work is licensed under the
Creative Commons Attribution License.

se, and because it is a case study to test our claim that an intersection types assignment system can be defined and that it is useful to formalize reasoning about the calculus itself.

Indeed the type system, which we present in the very last section of this work, has not been invented from scratch; on the contrary, it has been constructed moving from the domain equation $D = D \to \mathbb{S}D$, where $\mathbb{S}$ is (a variant of) the state monad. The method we follow is to solve such equation in the category of ω-algebraic lattices, whose objects and morphisms can be described via intersection type theories, that are the "logic" of such domains in the sense of [1]: see also [3]. The solution is constructed via the description of compact points of such a D, implicitly mirroring the inverse limit construction, but in a conceptually simpler way, consisting of an inductive definition of types and of their preorders. Once this has been obtained, subtyping and typing rules are derived in a uniform way, leading to a type assignment system which enjoys the "type-semantics" property, namely that denotational semantics of terms is fully characterized by their typings in the system. Soundness and completeness of the type system now follow straightforwardly.

Summary and results. In section 2 we outline the syntax of the untyped imperative λ-calculus λ_{imp}, and stress the difference with other calculi in literature. In section 3 we deal with solving the domain equation thoroughly. To do this we recall the concept of computational monad, then we consider the state and partiality monad $\mathbb{S}$. In the same section, we tackle the solution of the domain equation by breaking the circularity that arises out of the definition of the monad $\mathbb{S}$ itself. We conclude the section by modelling algebraic operators over the monad $\mathbb{S}$ and formalizing the model of λ_{imp}. In section 4, we solve the domain equation by inductively constructing type theories that induce a filter model.

Section 5 explains how to derive the type assignment system from the filter-model construction. We conclude establishing the type semantics theorem for our calculus. Finally, section 6 is devoted to the discussion of our results and to related works.

We assume familiarity with λ-calculus and intersection types; a comprehensive reference is [7] Part III. Some previous knowledge about computational monads and algebraic effects is useful: see e.g. [8] and [17] chap. 3.

2 An untyped imperative λ-calculus: λ_{imp}

Imperative extensions of the λ-calculus, both typed and type-free, are usually based on the call-by-value λ-calculus, and enriched with constructs for reading and writing to the store. Aiming at exploring the semantics of side effects in computational calculi, where "impure" functions are modeled by pure ones sending values to computations in the sense of [22], we consider the calculus introduced in [14], to which we add syntax denoting algebraic effect operations à la Plotkin and Power [23, 24, 25] over a suitable state monad.

Let $\mathbb{L} = \{\ell_0, \ell_1, \ldots\}$ be a denumerable set of abstract *locations*. Borrowing notation from [17], chap. 3, we consider denumerably many operator symbols get_ℓ and set_ℓ, obtaining:

Definition 2.1. *The syntax of the untyped imperative λ-calculus λ_{imp} has two sorts of terms, which are defined by mutual induction as follows:*

$$\begin{array}{rrcl} Val: & V,W & ::= & x \mid \lambda x.M \\ Com: & M,N & ::= & [V] \mid M \star V \mid get_\ell(\lambda x.M) \mid set_\ell(V,M) \qquad (\ell \in \mathbb{L}) \end{array}$$

Terms are of either sorts *Val* and *Com*, representing values and computations, respectively. The variable x is bound in $\lambda x.M$ and $get_\ell(\lambda x.M)$; terms are identified up to renaming of bound variables so that the

capture avoiding substitution $M[V/x]$ is always well defined; $FV(M)$ denotes the set of free variables in M. We call Val^0 the subset of closed $V \in Val$; similarly for Com^0.

Terms of the shape $[V]$ and $M \star V$ are written `return` V and M `>>=` V in Haskell-like syntax, respectively. With respect to [14] we write $[V]$ instead of *unit* V. With respect to the syntax of the "imperative λ-calculus" in [17], we do not have the *let* construct, nor the application VW among values. These constructs are indeed definable: $let\ x = M\ in\ N \equiv M \star (\lambda x.N)$ and $VW \equiv [W] \star V$ (where $\equiv$ is syntactic identity). In general, application among computations can be encoded by $MN \equiv M \star (\lambda z.\ N \star z)$, where z is fresh.

In a sugared notation from functional programming languages, we could have written location dereferentiation and assignment, respectively, as follows:

$$let\ x = !\ell\ in\ M \equiv get_\ell(\lambda x.M) \qquad \ell := V; M \equiv set_\ell(V, M)$$

Observe that we do not consider locations as values; consequently they cannot be dynamically created like with the **ref** operator from ML, nor is it possible to model aliasing. On the other hand, since the calculus is untyped, "strong updates" are allowed (differently than [9]). By this, we mean that when evaluating $set_\ell(V, M)$, the value V to which the location ℓ is updated bears no relation to the previous value, say W, of ℓ in the store: indeed, in our type assignment system W and V may well have completely different types.

3 Denotational semantics

We illustrate a denotational semantics of the calculus in the category of domains. The model is based on the solution of the domain equation:

$$D = [D \to [S \to (D \times S)_\perp]] \tag{1}$$

where S is a suitable space of stores over D, and $\mathbb{S}D = [S \to (D \times S)_\perp]$ is a variant of the state monad in [22], which is called the partiality and state monad in [13].

Before embarking on solving the equation (1), let us recall definitions of monads, of $\mathbb{S}$, and algebraic operators, and fix notations.

The partiality and state monad. We recall Wadler's type-theoretic definition of monads [26, 27], that is at the basis of their successful implementation in Haskell language, a natural interpretation of the calculus is into a cartesian closed category (ccc), such that two families of combinators, or a pair of polymorphic operators called the "unit" and the "bind", exist satisfying the *monad laws*. In what follows, $\mathscr{C}$ will be a *concrete ccc*. Examples are the category **Dom** of Scott domains with continuous functions, and its full subcategory ω-**ALG** of algebraic lattices with a countable basis.

Definition 3.1. (Computational Monad [27] §3) *Let $\mathscr{C}$ be a concrete ccc. A* functional computational monad*, henceforth just* monad *over $\mathscr{C}$ is a triple $(T, unit, \star)$ where T is a map over the objects of $\mathscr{C}$, and $unit$ and $\star$ are families of morphisms*

$$unit_A : A \to TA \qquad\qquad \star_{A,B} : TA \times (TB)^A \to TB$$

such that, writing $\star_{A,B}$ as an infix operator and omitting subscripts:

$$\begin{array}{lrcl} \textit{Left unit}: & unit\ a \star f & = & f\ a \\ \textit{Right unit}: & m \star unit & = & m \\ \textit{Assoc}: & (m \star f) \star g & = & m \star \boldsymbol{\lambda}\, d.(f\ d \star g) \end{array}$$

where $\lambda\!\!\!^{-}\, d.(f\, d \star g)$ *is the lambda notation for* $d \mapsto f\, d \star g$.

Remark 3.2. *In case of concrete ccc, Def. 3.1 coincides with the notion of* Kleisli triple $(T, \eta, _^{\dagger})$ *(see e.g. [22], Def. 1.2). The equivalence is obtained by relating maps* η *and* $_^{\dagger}$ *to unit and* $\star$ *operators, as follows:*

$$unit_X = \eta_X \qquad\qquad a \star f = f^{\dagger}(a)$$

The first equality is just a notational variation of the coercion of values into computations. The latter, instead, establishes the connection between the $\star$ *operator with the map* $_^{\dagger}$*, also called* extension operator*: if* $f : X \to TY$ *then* $f^{\dagger} : TX \to TY$ *is the unique map such that* $f = f^{\dagger} \circ \eta_X$.

Now, let us look closer how monads model effectful computations. We focus on **Dom**, a full subcategory of pointed Dcpo's with Scott continuous functions as morphisms. Then TX is the domain, or the type, of computations with values in X. In general TX has a richer structure than X itself, modeling partial computations, exceptions, non-determinism etcetera, including side effects. The mapping $unit_X : X \to TX$ is interpreted as the trivial computation $unit_X(x)$ just returning the value x, and it is an embedding if T satisfies the requirement that all the $unit_X$ are monos. Now a function $f : X \to TY$ models an "impure" program P with input in X and output in Y via a pure function returning the computation $f(x) \in TY$.

To relate equation (1) to the monad $\mathbb{S}$ we have to be more precise about the domain S. We consider $FX = X^{\mathbb{L}}$ the domain of *stores* over X with locations in $\mathbb{L}$, namely the set $X^{\mathbb{L}}$, ranged over by ς, of maps from locations to points of X; FX is a domain with the order inherited from X. The map FX obviously extends to a locally continuous functor by setting $F(g) = g^{\mathbb{L}} = g \circ _ : X^{\mathbb{L}} \to Y^{\mathbb{L}}$ for any $g : X \to Y$.

We are now in place to properly define the partiality and state monad $\mathbb{S}$ in terms of Def. 3.1.

Definition 3.3. (Partiality and state monad) *Given the domain* $S = X^{\mathbb{L}}$ *representing a notion of state, we define the partiality and state monad* $(\mathbb{S}, unit\ , \star)$*, as the mapping*

$$\mathbb{S}X = [S \to (X \times S)_{\perp}]$$

where $(X \times S)_{\perp}$ *is the lifting of the cartesian product* $X \times S$*, equipped with two (families of) operators* $unit$ *and* $\star$ *defined as follows:*

$$\begin{array}{l} unit\ x ::= \lambda\!\!\!^{-}\, \varsigma.(x, \varsigma) \\ (c \star f)\varsigma = f^{\dagger}(c)(\varsigma) ::= \left\{ \begin{array}{ll} f(x)(\varsigma') & \text{if } c(\varsigma) = (x, \varsigma') \neq \perp \\ \perp & \text{if } c(\varsigma) = \perp \end{array} \right. \end{array}$$

where we omit subscripts.

Remark 3.4. *A function* $f : X \to \mathbb{S}Y$ *has type* $X \to S \to (Y \times S)_{\perp}$*, which is isomorphic to* $(X \times S) \to (Y \times S)_{\perp}$*; if* f *is the interpretation of an "imperative program" P, and it is the currying of* f'*, then we expect that* $f\, x\, \varsigma = f'(x, \varsigma) = (y, \varsigma')$ *if* y *and* ς' *are respectively the value and the final store obtained from the evaluation of* $P(x)$ *starting in* ς*, if it terminates;* $f\, x\, \varsigma = f'(x, \varsigma) = \perp$ *otherwise. Clearly* f' *is the familiar interpretation of the imperative program P as a state transformation mapping.*

A domain equation. Evidently, the domain FX depends on X itself; in contrast, while defining $\mathbb{S}X$ the domain S has to be fixed, since otherwise the definition of the $\star$ operator does not make sense, and $\mathbb{S}$ is not even a functor. A solution would be to take $S = FD$, where $D \cong [D \to \mathbb{S}D]$, but this is clearly circular.

To break the circularity, we define the mixed-variant bi-functor $G : \mathbf{Dom}^{op} \times \mathbf{Dom} \to \mathbf{Dom}$ by

$$G(X, Y) = [FX \to (Y \times FY)_{\perp}]$$

whose action on morphisms is illustrated by the diagram:

$$\begin{tikzcd}
FX' \arrow[r, "Ff"] \arrow[d, dashed, "G(f,g)(\alpha)"'] & FX \arrow[d, "\alpha"] \\
(Y' \times FY')_\perp & (Y \times FY)_\perp \arrow[l, "(g \times Fg)_\perp"]
\end{tikzcd}$$

where $f : X' \to X$, $g : Y \to Y'$ and $\alpha \in G(X,Y)$. Now it is routine to prove that G is locally continuous so that, by the inverse limit technique, we can find the initial solution to the domain equation (which is the same as Equation (1)):

$$D = [D \to G(D,D)] \tag{2}$$

In summary we have:

Theorem 3.5. *There exists a domain D such that the state monad $\mathbb{S}_D$ with state domain $S = [\mathbb{L} \to D]$ is a solution in **Dom** to the domain equation:*

$$D = [D \to \mathbb{S}_D D]$$

Moreover, it is initial among all solutions to such equation.

Proof. Take D to be the (initial) solution to Equation (2); now if $S = FD = D^{\mathbb{L}}$ then $\mathbb{S}_D D = G(D,D)$. □

Algebraic operations over $\mathbb{S}$. Monads are about composition of morphisms of some special kind. However, thinking of $f : X \to \mathbb{S}X$ as the meaning of a program with side effects doesn't tell anything about side effects themselves, that are produced by reading and writing values from and to stores in S.

To model effects associated to a monad, Plotkin and Power have proposed in [23, 24, 25] a theory of *algebraic operations*. A gentle introduction to the theory can be found in [17], chap. 3, from which we borrow the notation.

Suppose that T is a monad over a category $\mathscr{C}$ with terminal object $\mathbf{1}$ and all finite products; then an algebraic operation $\mathbf{op}$ with arity n is a family of morphisms $\mathbf{op}_X : (TX)^n \to TX$, where $(TX)^n = TX \times \cdots \times TX$ is the n-times product of TX with itself, such that

$$\mathbf{op}_X \circ \Pi_n f^\dagger = f^\dagger \circ \mathbf{op}_X \tag{3}$$

where $f : X \to TX$ and $\Pi_n f^\dagger = f^\dagger \times \cdots \times f^\dagger : (TX)^n \to (TX)^n$. In case of the concrete category **Dom**, $\mathbf{1}$ is a singleton and products are sets of tuples. Then $\mathbf{op}_X$ is an operation of arity n of an algebra with carrier TX; since $f^\dagger : TX \to TX$, we have that $(\Pi_n f^\dagger)\langle x_1, \ldots, x_n\rangle = \langle f^\dagger(x_1), \ldots, f^\dagger(x_n)\rangle$, and Equation (3) reads as:

$$\mathbf{op}_X(f^\dagger(x_1), \ldots, f^\dagger(x_n)) = f^\dagger(\mathbf{op}_X(x_1, \ldots, x_n))$$

namely the functions $f^\dagger$ are homomorphisms w.r.t. $\mathbf{op}_X$.

Algebraic operations $\mathbf{op}_X : (TX)^n \to TX$ do suffice in case T is, say, the nondeterminism or the output monad, but cannot model side effects operations in case of the store monad. This is because read and write operations implement a bidirectional action of stores to programs and of programs to stores. What is needed instead is the generalized notion of operation proposed in [23] (and further studied in

[18]); such construction, that is carried out in a suitable enriched category, can be instantiated to the case of **Dom** as follows:

$$\mathbf{op} : P \times (TX)^A \to TX \cong (TX)^A \to (TX)^P$$

where P is the domain of parameters and A of generalized arities, and the rightmost "type" is the interpretation in **Dom** of Def. 1 in [23]. For such operations Equation (3) is generalized as follows (see [17], Def. 13):

$$\mathbf{op}(p,k) \star f = f^\dagger(\mathbf{op}(p,k)) = \mathbf{op}(p, f^\dagger \circ k) = \mathbf{op}(p, \boldsymbol{\lambda}\, x.(k(x) \star f)) \tag{4}$$

where $f : X \to TX$, $p \in P$ and $k : A \to TX$.

Denotational semantics of terms. Given the monad $(\mathbb{S}, \eta, _^\dagger)$ and the domain D from Theorem 3.5, we first interpret the constants get_ℓ and set_ℓ as generalized operations over $\mathbb{S}$. By taking $P = \mathbf{1}$ and $A = D$ we define:

$$[\![get_\ell]\!] : \mathbf{1} \times (\mathbb{S}D)^D \to \mathbb{S}D \simeq (\mathbb{S}D)^D \to \mathbb{S}D \quad \text{by} \quad [\![get_\ell]\!]\, d\, \varsigma = d\,(\varsigma\,(\ell))\,\varsigma$$

where $d \in D$ is identified with its image in $D \to \mathbb{S}D$, and $\varsigma \in S = [\mathbb{L} \to D]$. On the other hand, taking $P = D$ and $A = \mathbf{1}$, we define:

$$[\![set_\ell]\!] : D \times (\mathbb{S}D)^{\mathbf{1}} \to \mathbb{S}D \simeq D \times \mathbb{S}D \to \mathbb{S}D \quad \text{by } [\![set_\ell]\!](d,c)\,\varsigma = c(\varsigma[\ell \mapsto d])$$

where $c \in \mathbb{S}D = S \to (D \times S)_\perp$ and $\varsigma[\ell \mapsto d]$ is the store sending ℓ to d and it is equal to ς otherwise.

Then we interpret values from *Val* in D and computations from *Com* in $\mathbb{S}D$. More precisely we define the maps $[\![\cdot]\!]^D : Val \to Env \to D$ and $[\![\cdot]\!]^{\mathbb{S}D} : Com \to Env \to \mathbb{S}D$, where $Env = Var \to D$ is the set of environments interpreting term variables in *Var*:

Definition 3.6. *A λ_{imp}-**model** is a structure $\mathscr{D} = (D, \mathbb{S}, [\![\cdot]\!]^D, [\![\cdot]\!]^{\mathbb{S}D})$ such that:*

1. *D is a domain s.t. $D \cong D \to \mathbb{S}D$ via (Φ, Ψ), where $\mathbb{S}$ is the partiality and state monad of stores over D;*
2. *for all $e \in Env$, $V \in Val$ and $M \in Com$:*

$$\begin{array}{ll}
[\![x]\!]^D e = e(x) & [\![\lambda x.M]\!]^D e = \Psi(\boldsymbol{\lambda}\, d \in D.\, [\![M]\!]^{\mathbb{S}D} e[x \mapsto d]) \\
[\![\,[V]\,]\!]^{\mathbb{S}D} e = unit\ ([\![V]\!]^D e) & [\![M \star V]\!]^{\mathbb{S}D} e = ([\![M]\!]^{\mathbb{S}D} e) \star \Phi([\![V]\!]^D e) \\
[\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e = [\![get_\ell]\!]\, \Phi([\![\lambda x.M]\!]^D e) & [\![set_\ell(V,M)]\!]^{\mathbb{S}D} e = [\![set_\ell]\!]([\![V]\!]^D e, [\![M]\!]^{\mathbb{S}D} e)
\end{array}$$

By unravelling definitions and applying to an arbitrary store $\varsigma \in S$, the last two clauses can be written:

$$\begin{array}{rcl}
[\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e\,\varsigma & = & [\![M]\!]^{\mathbb{S}D}(e[x \mapsto \varsigma(\ell)])\,\varsigma \\
[\![set_\ell(V,M)]\!]^{\mathbb{S}D} e\,\varsigma & = & [\![M]\!]^{\mathbb{S}D} e\,(\varsigma[\ell \mapsto [\![V]\!]^D e])
\end{array}$$

We say that the equation $M = N$ is *true* in $\mathscr{D}$, written $\mathscr{D} \models M = N$, if $[\![M]\!]^{\mathbb{S}D} e = [\![N]\!]^{\mathbb{S}D} e$ for all $e \in Env$.

Proposition 3.7. *The following equations are true in $\mathscr{D}$:*

1. $[V] \star (\lambda x.M) = M[V/x]$
2. $M \star \lambda x.[x] = M$
3. $(L \star \lambda x.M) \star \lambda y.N = L \star \lambda x.(M \star \lambda y.N)$
4. $get_\ell(\lambda x.M) \star W = get_\ell(\lambda x.(M \star W))$

5. $set_\ell(V,M) \star W = set_\ell(V, M \star W)$

where $x \notin FV(\lambda y.N)$ in (3) and $x \notin FV(W)$ in (4).

Proof. By definition and straightforward calculations. For example, to see (4), let $\varsigma \in S$ be arbitrary and $e' = e[x \mapsto \varsigma(\ell)]$; then, omitting the apices of the interpretation mappings $[\![\cdot]\!]$:

$$[\![get_\ell(\lambda x.M) \star W]\!] e\, \varsigma = ([\![W]\!]e)^\dagger([\![M]\!]e')\, \varsigma$$

On the other hand:

$$[\![get_\ell(\lambda x.(M \star W))]\!] e\, \varsigma = ([\![\lambda x.(M \star W)]\!]e)\, \varsigma(\ell)\, \varsigma = [\![M \star W]\!]e'\, \varsigma = ([\![W]\!]e')^\dagger([\![M]\!]e')\, \varsigma$$

But $x \notin FV(W)$ implies $[\![W]\!]e' = [\![W]\!]e$, and we are done. □

Equations (1)-(3) in Prop. 3.7 are the *monadic laws* from Def. 3.1, equations (4)-(5) are instances of Equation (4).

4 The filter-model construction

In this section we switch to the category ω-**ALG** of ω-algebraic lattices. ω-**ALG** is the full subcategory of **Dom** of complete lattices $(D, \sqsubseteq_D)$ such that the set $K(D)$ of compact points is countable and $d = \bigsqcup\{e \in K(D) \mid e \sqsubseteq_D d\}$ is a directed sup for all $d \in D$.

Observe that morphisms in ω-**ALG** are Scott-continuous maps, hence preserving direct sups, but not necessary arbitrary ones.

Definition 4.1. *An* intersection type theory*, shortly* itt*, is a pair $Th_A = (\mathscr{L}_A, \leq_A)$ where $\mathscr{L}_A$, the* language *of Th_A, is a countable set of type expressions closed under $\wedge$, and $\omega_A \in \mathscr{L}_A$ is a special constant; $\leq_A$ is a pre-order over $\mathscr{L}_A$ closed under the following rules:*

$$\alpha \leq_A \omega_A \qquad \alpha \wedge \beta \leq_A \alpha \qquad \alpha \wedge \beta \leq_A \beta \qquad \frac{\alpha \leq_A \alpha' \quad \beta \leq_A \beta'}{\alpha \wedge \beta \leq_A \alpha' \wedge \beta'}$$

In the literature, the operator $\wedge$ is called *intersection*, and ω_A the *universal type*. Informally, Th_A is identified with the set of inequalities $\alpha \leq_A \beta$, so that, abusing terminology, we shall also say that $\leq_A$ is a type theory. Clearly, the quotient $\mathscr{L}_A/_{\leq_A}$ is an inf-semilattice, with $\wedge$ as meet and ω_A as top element.

Definition 4.2. *A non empty $F \subseteq \mathscr{L}_A$ is a* filter *of Th_A if it is closed under $\wedge$ and upward closed w.r.t. $\leq_A$; let $\mathscr{F}_A$ be the set of filters of Th_A. The* principal filter over α *is the filter $\uparrow_A \alpha = \{\beta \in \mathscr{L}_A \mid \alpha \leq_A \beta\}$.*

The following proposition is the fundamental fact about intersection types and ω-algebraic lattices:

Proposition 4.3 (Representation theorem)**.** *The partial order $(\mathscr{F}_A, \subseteq)$ of the filters of an intersection type theory Th_A is an ω-algebraic lattice, whose compact elements are the principal filters. Vice versa, any ω-algebraic lattice A is isomorphic the poset $(\mathscr{F}_A, \subseteq)$ of filters of some intersection type theory Th_A.*

Proof. (Sketch). The first part is established via a direct argument. To the second part, let $\mathscr{L}_A = \{\alpha_d \mid d \in K(A)\}$ (where each α_d is a new type constant) which is countable by hypothesis, and take $Th_A = \{\alpha_d \leq_A \alpha_e \mid e \sqsubseteq d\}$. By observing that $\alpha_d \wedge_A \alpha_e = \alpha_{d \sqcup e}$, where $d \sqcup e \in K(A)$ if $d, e \in K(A)$, and that $\omega_A = \alpha_\top$, we have that Th_A is an intersection type theory and that $\mathscr{L}_A/_{\leq_A}$ is isomorphic to $K^{op}(A)$ ordered by the opposite to the ordering of A. From this it follows that $\mathscr{F}_A$ is isomorphic to the ideal completion of $K(A)$, which in turn is isomorphic to A by algebraicity. □

The above proposition substantiates the claim that intersection type theories are the *logic* of the domains in ω-**ALG**. The theory Th_A is called the Lindenbaum algebra of A in [1]; Prop. 4.3 is the object part of the construction establishing that intersection type theories, together with a suitable notion of morphisms among them, do form a category that is equivalent to ω-**ALG**; such equivalence is an instance of Stone duality as studied in [1] w.r.t. the category of 2/3 SFP domains, when ω-**ALG** is viewed as a subcategory of topological spaces: see e.g. [3].

The second part in the proof of Prop. 4.3 is the most relevant to us: it provides a recipe to describe a domain via a formal system, deriving inequalities among type expressions that encode "finite approximations" of points in a domain, hence of the denotations of terms if the domain is a λ_{imp}-model. Therefore, we seek theories Th_D and Th_S such that:

$$\mathscr{F}_D \cong [\mathscr{F}_D \to [\mathscr{F}_S \to (\mathscr{F}_D \times \mathscr{F}_S)_\perp]] \tag{5}$$

The first step is to show how the functors involved in the equation above, namely the lifting, the product and the (continuous) function space, of which $S = [\mathbb{L} \to D]$ is a special case, can be put in correspondence with the construction of new theories out of the theories which determine the domains combined by the functors. As this is known after [1], we just recall their definitions, and fix the notation.

Definition 4.4. *Suppose that the theories Th_A and Th_B are given, then for $\alpha \in \mathscr{L}_A$ and $\beta \in \mathscr{L}_B$ define:*

$$\begin{array}{llll} \mathscr{L}_{A_\perp} & \psi & ::= & \alpha \mid \psi \wedge \psi' \mid \omega_{A_\perp} \\ \mathscr{L}_{A\to B} & \phi & ::= & \alpha \to \beta \mid \phi \wedge \phi' \mid \omega_{A\to B} \end{array} \qquad \begin{array}{llll} \mathscr{L}_{A\times B} & \pi & ::= & \alpha \times \beta \mid \pi \wedge \pi' \mid \omega_{A\times B} \\ \mathscr{L}_{stA} & \sigma & ::= & \langle \ell : \alpha \rangle \mid \sigma \wedge \sigma' \mid \omega_S \end{array}$$

Then, we define the following sets of axioms:

1. $\alpha \leq_A \alpha' \Longrightarrow \alpha \leq_{A_\perp} \alpha'$.
2. $\omega_{A\times B} \leq_{A\times B} \omega_A \times \omega_B$ *and all instances of* $(\alpha \times \beta) \wedge (\alpha' \times \beta') \leq_{A\times B} (\alpha \wedge \alpha') \times (\beta \wedge \beta')$.
3. $\omega_{A\to B} \leq_{A\to B} \omega_A \to \omega_B$ *and all instances of* $(\alpha \to \beta) \wedge (\alpha \to \beta') \leq_{A\to B} \alpha \to (\beta \wedge \beta')$.
4. $\omega_S \leq_{stA} \langle \ell : \omega_A \rangle$ *and all instances of* $\langle \ell : \alpha \rangle \wedge \langle \ell : \alpha' \rangle \leq_{stA} \langle \ell : \alpha \wedge \alpha' \rangle$.

Finally, the theories $Th_{A\times B}$, $Th_{A\to B}$ and Th_{stA} are respectively closed under the rules:

$$\frac{\alpha \leq_A \alpha' \quad \beta \leq_B \beta'}{\alpha \times \beta \leq_{A\times B} \alpha' \times \beta'} \qquad \frac{\alpha' \leq_A \alpha \quad \beta \leq_B \beta'}{\alpha \to \beta \leq_{A\to B} \alpha' \to \beta'} \qquad \frac{\alpha \leq_A \alpha'}{\langle \ell : \alpha \rangle \leq_{stA} \langle \ell : \alpha' \rangle}$$

If $X \in \mathscr{F}_{A\to B}$ and $Y \in \mathscr{F}_A$, where either $A = D$ and $B = \mathbb{S}D$ or $A = S$ and $B = (D \times S)_\perp$, we define:

$$X \cdot Y = \{\psi \mid \exists \varphi \in Y.\ \varphi \to \psi \in X\}$$

We write $\varphi =_A \psi$ if $\varphi \leq_a \psi \leq_A \varphi$. For a map $f : \mathscr{F}_A \to \mathscr{F}_B$, define

$$\Lambda(f) = \uparrow_{A\to B} \{\varphi \to \psi \in \mathscr{L}_{A\to B} \mid \psi \in f(\uparrow_A \varphi)\}$$

Lemma 4.5. *If $X \in \mathscr{F}_{A\to B}$ and $Y \in \mathscr{F}_A$ then $X \cdot Y \in \mathscr{F}_B$. Moreover the map $_ \cdot _$ is continuous in both its arguments.*

Proof. We have $X \ni \omega_{A\to B} \leq \omega_A \to \omega_B$ and that $\omega_A \in Y$, hence $\omega_B \in X \cdot Y$. Since $\to$ is monotonic in its second argument, $X \cdot Y$ is upward closed. Finally, if $\psi_1, \psi_2 \in X \cdot Y$ then $\varphi_i \in Y$ and $\varphi_i \to \psi_i \in X$ for $i = 1, 2$ and some φ_1, φ_2; then, by antimonotonicity of $\to$ w.r.t. its first argument, $\varphi_i \to \psi_i \leq \varphi_1 \wedge \varphi_2 \to \psi_i \in X$

being X upward closed, and $\varphi_1 \wedge \varphi_2 \to \psi_1 \wedge \psi_2 = (\varphi_1 \wedge \varphi_2 \to \psi_1) \wedge (\varphi_1 \wedge \varphi_2 \to \psi_2) \in X$ since X is closed under intersections.

Concerning continuity, we have to show that $X \cdot Y = \bigsqcup \mathscr{Z}$ where $\mathscr{Z} = \{\uparrow \varphi \cdot \uparrow \psi \mid \varphi \in X \ \& \ \psi \in Y\}$, and the sup of a family of filters is the least filter including the union of the family. Inclusion from left to right follows by observing that if $\chi \in X \cdot Y$ then $\psi \to \chi \in X$ for some $\psi \in Y$, and of course $\chi \in \uparrow(\psi \to \chi) \cdot \uparrow \psi$. Viceversa if $\chi \in \bigsqcup \mathscr{Z}$ then for finitely many ψ_i, χ_i we have that $\bigcap_i \chi_i \leq \chi$, $\psi_i \in Y$ and $\psi_i \to \chi_i \in X$. It follows that $\chi_i \in X \cdot Y$ for all i, and hence the thesis since $X \cdot Y$ is a filter by the above. □

Lemma 4.6. *If $f : \mathscr{F}_A \to \mathscr{F}_B$ is continuous then $\Lambda(f) \in \mathscr{F}_{A \to B}$. $\Lambda(\cdot)$ is itself continuous and such that:*

$$\Lambda(f) \cdot X = f(X) \qquad \Lambda(\lambda\!\!\lambda Y.(X \cdot Y)) = X$$

Proof. Easy by unfolding definitions and by Lemma 4.5. □

Now we can establish:

Proposition 4.7. *The following are isomorphisms in* ω-**ALG***:*

$$\mathscr{F}_{A_\perp} \cong (\mathscr{F}_A)_\perp, \quad \mathscr{F}_{A \times B} \cong \mathscr{F}_A \times \mathscr{F}_B, \quad \mathscr{F}_{A \to B} \cong [\mathscr{F}_A \to \mathscr{F}_B] \quad \textit{and} \quad \mathscr{F}_{stA} \cong \mathbb{L} \to \mathscr{F}_A.$$

Proof. That $\mathscr{F}_{A_\perp} \cong (\mathscr{F}_A)_\perp$ is a consequence of the fact that $\omega_A <_{A_\perp} \omega_{A_\perp}$ is strict, hence $\uparrow \omega_{A_\perp}$ is the new bottom added to $\mathscr{F}_A$. $\mathscr{F}_{A \times B} \cong \mathscr{F}_A \times \mathscr{F}_B$ is induced by the continuous extension of the map $\uparrow(\alpha \times \beta) \mapsto (\uparrow \alpha, \uparrow \beta)$, that is clearly invertible. Finally, $\mathscr{F}_{A \to B} \cong [\mathscr{F}_A \to \mathscr{F}_B]$ is Lemma 4.6, and $\mathscr{F}_{stA} \cong [\mathbb{L} \to \mathscr{F}_A]$ can be reduced to a particular case of the latter by using the continuous extension of the map $\uparrow \langle \ell : \delta \rangle \mapsto (\ell \mapsto \uparrow \delta)$. □

The next step is to apply Prop. 4.7 to describe the compact elements of $D \cong \mathscr{F}_D$ and of $S \cong \mathscr{F}_S$ and the (inverse of) their orderings. Alas, this cannot be done directly because of the recursive nature of the equation (5), but it can be obtained by mirroring the inverse limit construction, e.g. along the lines of [3, 5]. Although possible in principle, such a construction requires lots of machinery from the theory of the solution of domain equations; instead we follow the shorter path to define the involved type theories below simply by mutual induction:

Definition 4.8. *Let us abbreviate $S = D^{\mathbb{L}}$ and $C = (D \times S)_\perp$; then define the following itt languages by mutual induction:*

$$\begin{array}{llllllllll}
\mathscr{L}_D: & \delta & ::= & \delta \to \tau \mid \delta \wedge \delta' \mid \omega_D & \qquad & \mathscr{L}_S: & \sigma & ::= & \langle \ell : \delta \rangle \mid \sigma \wedge \sigma' \mid \omega_S \\
\mathscr{L}_C: & \kappa & ::= & \delta \times \sigma \mid \kappa \wedge \kappa' \mid \omega_C & \qquad & \mathscr{L}_{\mathbb{S}D}: & \tau & ::= & \sigma \to \kappa \mid \tau \wedge \tau' \mid \omega_{\mathbb{S}D}
\end{array}$$

We assume that $\wedge$ and $\times$ take precedence over $\to$ and that $\to$ associates to the right so that $\delta \to \tau \wedge \tau'$ reads as $\delta \to (\tau \wedge \tau')$ and $\delta' \to \sigma' \to \delta'' \times \sigma''$ reads as $\delta' \to (\sigma' \to (\delta'' \times \sigma''))$.

Then the relations $\leq_D, \leq_S, \leq_C$ and $\leq_{\mathbb{S}D}$ are the least itt's over the respective languages satisfying the following axioms:

$$(\delta \to \tau) \wedge (\delta \to \tau') \leq_D \delta \to \tau \wedge \tau' \qquad \omega_D \leq_D \omega_D \to \omega_{\mathbb{S}D}$$

$$\langle \ell : \delta \rangle \wedge \langle \ell : \delta' \rangle \leq_S \langle \ell : \delta \wedge \delta' \rangle \qquad \omega_S \leq_S \langle \ell : \omega_D \rangle$$

$$(\delta \times \sigma) \wedge (\delta' \times \sigma') \leq_C (\delta \wedge \delta') \times (\sigma \wedge \sigma')$$

$$(\sigma \to \kappa) \wedge (\sigma \to \kappa') \leq_{\mathbb{S}D} \sigma \to \kappa \wedge \kappa' \qquad \omega_{\mathbb{S}D} \leq_{\mathbb{S}D} \omega_S \to \omega_C$$

and closed under the rules:

$$\frac{\delta' \leq_D \delta \quad \tau \leq_{\mathbb{S}D} \tau'}{\delta \to \tau \leq_D \delta' \to \tau'} \qquad \frac{\delta \leq_D \delta'}{\langle \ell : \delta \rangle \leq_S \langle \ell : \delta' \rangle} \qquad \frac{\delta \leq_D \delta' \quad \sigma \leq_S \sigma'}{\delta \times \sigma \leq_C \delta' \times \sigma'} \qquad \frac{\sigma' \leq_S \sigma \quad \kappa \leq_C \kappa'}{\sigma \to \kappa \leq_{\mathbb{S}D} \sigma' \to \kappa'}$$

Remark 4.9. *Observe that the only constants used in Def. 4.8 are the ω's; also we have plenty of equivalences $\varphi = \psi$, namely relations $\varphi \leq \psi \leq \varphi$, involving these constants, that are induced by the definition of the itt's above. For example $\delta \to \omega_{\mathbb{S}D} = \omega_D$ is derivable since $\omega_D \leq_D \delta \to \omega_{\mathbb{S}D} \leq_D \omega_D$ are axioms of Th_D; similarly $\langle \ell : \omega_D \rangle = \omega_S$ and $\omega_S \to \omega_C = \omega_{\mathbb{S}D}$.*

However, none of the theories above is trivial, because $\omega_C \not\leq_C \omega_D \times \omega_S$. This is implied by $\uparrow \omega_C \subset \uparrow (\omega_D \times \omega_S)$, that is $\uparrow \omega_C$ corresponds to the new bottom element added to $\mathscr{F}_{D\times S} \cong \mathscr{F}_D \times \mathscr{F}_S$ in $\mathscr{F}_C = \mathscr{F}_{(D\times S)_\perp} \cong (\mathscr{F}_{D\times S})_\perp$.

The choice of not having atomic types α in $\mathscr{L}_D$ is minimalistic, and parallels the analogous definition 5.2.1 of [2], where the only constant in the domain logic of a lazy λ-model $D \cong [D \to D]_\perp$ is t (true), corresponding to our ω_D, and having $t \not\leq (t \to t)_\perp$ in the theory.

As a more careful description of $\mathscr{F}_D$ would show, by relating its construction to the solution of eq. (2) in Theorem 3.5, the reason why the ω's suffice is that the domain equation has a non trivial initial solution. Adding atomic types α to $\mathscr{L}_D$ also leads to a filter model $\mathscr{F}_{D'}$ of λ_{imp}, which is however not isomorphic to $[\mathscr{F}_{D'} \to \mathbb{S}(\mathscr{F}_{D'})]$, which is what we show to hold for $\mathscr{F}_D$ in the next theorem. To restore the desired isomorphism it suffices to add axioms $\alpha =_{D'} \omega_{D'} \to \omega_S \to (\alpha \times \omega_S)$ for all atomic α: these correspond to the axioms $\alpha = \omega \to \alpha$ in [6], which are responsible of obtaining a "natural equated" solution to the equation $D = [D \to D]$ of Scott's model: see [5].

Theorem 4.10. *The theories Th_D and Th_S induce the filter domains $\mathscr{F}_D$ and $\mathscr{F}_S$ which satisfy equation (5).*

Proof. By Prop. 4.7, $\mathscr{F}_{D\to\mathbb{S}D} \cong [\mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}] \cong [\mathscr{F}_D \to [\mathscr{F}_S \to (\mathscr{F}_D \times \mathscr{F}_S)_\perp]]$; then the thesis follows since $\mathscr{F}_D = \mathscr{F}_{D\to\mathbb{S}D}$ by construction. □

The λ_{imp} filter model. According to Def. 3.6, to show that $\mathscr{F}_D$ is a λ_{imp}-model it remains to see that $\mathscr{F}_{\mathbb{S}D} \cong \mathbb{S}\mathscr{F}_D$ can be endowed with the structure of a monad, which amounts to say that the maps $unit^{\mathscr{F}} : \mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}$ and $\star^{\mathscr{F}} : \mathscr{F}_{\mathbb{S}D} \times [\mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}] \to \mathscr{F}_{\mathbb{S}D}$ are definable in such a way to satisfy the monadic laws. This easily follows by instantiating the definition of the state monad Def. 3.3 to the subcategory of filter domains:

$$unit^{\mathscr{F}} X =_{def} \Lambda(\boldsymbol{\lambda}\, Y \in \mathscr{F}_S.(X,Y)) = \texttt{Filt}\{\sigma \to \delta \times \sigma \in \mathscr{L}_{\mathbb{S}D} \mid \delta \in X\} \tag{6}$$

where $\texttt{Filt}\,U$ is the least filter including the set U. On the other hand, observing that $\mathscr{F}_{\mathbb{S}D} \times [\mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}] \to \mathscr{F}_{\mathbb{S}D} \cong \mathscr{F}_{\mathbb{S}D} \times \mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}$, for all $X \in \mathscr{F}_{\mathbb{S}D}$, $Y \in \mathscr{F}_D$ and $Z \in \mathscr{F}_S$ we expect that:

$$(X \star^{\mathscr{F}} Y) \cdot Z = let\ (U,V) = X \cdot Z\ in\ (Y \cdot U) \cdot V = \begin{cases} (Y \cdot U) \cdot V & \text{if } X \cdot Z = U \times V \neq \uparrow_C \omega_C \\ \uparrow_C \omega_C & \text{otherwise} \end{cases}$$

Hence we put

$$X \star^{\mathscr{F}} Y =_{def} \texttt{Filt}\{\sigma \to \delta'' \times \sigma'' \in \mathscr{L}_{\mathbb{S}D} \mid \exists \delta', \sigma'.\ \sigma \to \delta' \times \sigma' \in X\ \&\ \delta' \to \sigma' \to \delta'' \times \sigma'' \in Y\} \cup \uparrow_C \omega_C \tag{7}$$

The final step is to define the interpretations of $get_\ell : (\mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}) \to \mathscr{F}_{\mathbb{S}D} \cong \mathscr{F}_D \to \mathscr{F}_{\mathbb{S}D}$ and $set_\ell : \mathscr{F}_D \times \mathscr{F}_{\mathbb{S}D} \to \mathscr{F}_{\mathbb{S}D}$, which also are derivable from their interpretation into a generic model $\mathscr{D}$. Indeed, according to Def. 3.6, for any $X \in \mathscr{F}_D$ and $Y \in \mathscr{F}_S$ we must have:

$$get_\ell(X) \cdot Y = X \cdot (Y \cdot \{\ell\}) \cdot Y$$

where we assume that $_ \cdot _$ associates to the left, and we abbreviate $Y \cdot \{\ell\} = \{\delta \in \mathscr{L}_D \mid \langle \ell : \delta \rangle \in Y\}$ representing the application of the "store" Y to the location ℓ. Now, let $Z =_{def} \{\tau \in \mathscr{L}_{\mathbb{S}D} \mid \exists \delta \in Y \cdot \{\ell\}.\ \delta \to \tau \in X\}$ then

$$X \cdot (Y \cdot \{\ell\}) \cdot Y = Z \cdot Y$$

If $\tau = \omega_{\mathbb{S}D}$ then $\delta \to \tau = \omega_D$, which trivially belongs to any filter; if instead $\tau \neq \omega_{\mathbb{S}D}$ then $\tau = \bigwedge_I \sigma_i \to \kappa_i$ and $\delta \to \tau \in X$ if and only if $\delta \to \sigma_i \to \kappa_i \in X$ for all $i \in I$. From this we conclude that

$$Z \cdot Y = \{\kappa \in \mathscr{L}_C \mid \exists \sigma \in Y \mid \sigma \to \kappa \in Z\} = \{\kappa \in \mathscr{L}_C \mid \exists \langle \ell : \delta \rangle \wedge \sigma \in Y.\ \delta \to \sigma \to \kappa \in X\}$$

and therefore the appropriate definition of $get_\ell(X)$ is

$$get_\ell(X) =_{def} \texttt{Filt}\{(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in \mathscr{L}_{\mathbb{S}D} \mid \delta \to (\sigma \to \kappa) \in X\} \tag{8}$$

Similarly, again by Def. 3.6, for any $X \in \mathscr{F}_D$, $Y \in \mathscr{F}_{\mathbb{S}D}$ and $Z \in \mathscr{F}_S$ we expect:

$$set_\ell^{\mathscr{F}}(X,Y) \cdot Z = Y \cdot (Z[\ell \mapsto X])$$

where $Z[\ell \mapsto X]$ is supposed to represent the update of Z by associating X to ℓ, namely:

$$Z[\ell \mapsto X] = \texttt{Filt}\{\{\langle \ell : \delta \rangle \mid \delta \in X\} \cup \{\langle \ell' : \delta' \rangle \mid \langle \ell' : \delta' \rangle \in Z \ \ \& \ \ \ell' \neq \ell\}\}$$

Then

$$\begin{aligned} Y \cdot (Z[\ell \mapsto X]) &= \texttt{Filt}\{\kappa \mid \exists \sigma \to \kappa \in Z[\ell \mapsto X].\ \sigma \to \kappa \in Y\} \\ &= \texttt{Filt}\{\kappa \mid \exists \sigma' \in Z, \delta \in X.\ \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in Y \ \ \& \ \ \ell \notin dom(\sigma')\} \end{aligned}$$

and therefore we define:

$$set_\ell^{\mathscr{F}}(X,Y) =_{def} \texttt{Filt}\{\sigma' \to \kappa \in \mathscr{L}_{\mathbb{S}D} \mid \exists \delta \in X.\ \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in Y \ \ \& \ \ \ell \notin dom(\sigma')\} \tag{9}$$

Eventually, by construction we have:

Theorem 4.11. *The structure $\mathscr{F} = (\mathscr{F}_D, \mathbb{S}, [\![\cdot]\!]^{\mathscr{F}_D}, [\![\cdot]\!]^{\mathscr{F}_{\mathbb{S}D}})$ is a λ_{imp}-model where, for $e : Var \to \mathscr{F}_D$ the interpretations $[\![V]\!]^{\mathscr{F}_D} e \in \mathscr{F}_D$ and $[\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e \in \mathscr{F}_{\mathbb{S}D}$ are defined by:*

$$\begin{aligned} [\![x]\!]^{\mathscr{F}_D} e &= e(x) & [\![\lambda x.M]\!]^{\mathscr{F}_D} e &= \Lambda(\boldsymbol{\lambda} X \in \mathscr{F}_D.\ [\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e[x \mapsto X]) \\ [\![[V]]\!]^{\mathscr{F}_{\mathbb{S}D}} e &= unit^{\mathscr{F}}([\![V]\!]^{\mathscr{F}_D} e) & [\![M \star V]\!]^{\mathscr{F}_{\mathbb{S}D}} e &= ([\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e) \star^{\mathscr{F}} ([\![V]\!]^{\mathscr{F}_D} e) \\ [\![get_\ell(\lambda x.M)]\!]^{\mathscr{F}_{\mathbb{S}D}} e &= get_\ell([\![\lambda x.M]\!]^{\mathscr{F}_D} e) & [\![set_\ell(V,M)]\!]^{\mathscr{F}_{\mathbb{S}D}} e &= set_\ell^{\mathscr{F}}([\![V]\!]^{\mathscr{F}_D}, [\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e) \end{aligned}$$

Proof. By applying equations (6)-(9) to the clauses of Def. 3.6. □

5 Deriving the type assignment system

The definition of a type assignment system can be obtained out of the construction of the filter-model. The system is nothing else than the description of the denotation of terms in the particular λ_{imp}-model $\mathscr{F}$, which is possible because both $[\![V]\!]^{\mathscr{F}_D}e \in \mathscr{F}_D$ and $[\![M]\!]^{\mathscr{F}_{\mathbb{S}D}}e \in \mathscr{F}_{\mathbb{S}D}$, and filters are sets of types.

Types are naturally interpreted as subsets of the domains of term interpretation, namely $\mathscr{F}_D$ or $\mathscr{F}_{\mathbb{S}D}$ according to their kind; by applying to the present case the same construction as in [6] and [1], which originates from Stone duality for boolean algebras, we set:

Definition 5.1. *For $A = D, S, C$ and $\mathbb{S}D$ and $\varphi \in \mathscr{L}_A$ define:*

$$[\![\varphi]\!]^{\mathscr{F}} = \{X \in \mathscr{F}_A \mid \varphi \in X\}$$

Such interpretation is a generalization of the natural interpretation of intersection types over an extended type structure [12], which in the present case yields:

Proposition 5.2. *For $A = D, S, C$ and $\mathbb{S}D$ and $\varphi, \psi \in \mathscr{L}_A$ we have:*

$$[\![\varphi \wedge \psi]\!]^{\mathscr{F}} = [\![\varphi]\!]^{\mathscr{F}} \cap [\![\psi]\!]^{\mathscr{F}}, \qquad [\![\omega_A]\!]^{\mathscr{F}} = \mathscr{F}_A \quad \textit{and} \quad \varphi \leq \psi \Longrightarrow [\![\varphi]\!]^{\mathscr{F}} \subseteq [\![\psi]\!]^{\mathscr{F}}$$

Moreover:

1. $[\![\delta \to \tau]\!]^{\mathscr{F}} = \{X \in \mathscr{F}_D \mid \forall Y \in [\![\delta]\!]^{\mathscr{F}}.\, X \cdot Y \in [\![\tau]\!]^{\mathscr{F}}\}$
2. $[\![\langle \ell : \delta \rangle]\!]^{\mathscr{F}} = \{X \in \mathscr{F}_S \mid X \cdot \{\ell\} \in [\![\delta]\!]^{\mathscr{F}}\}$
3. $[\![\delta \times \sigma]\!]^{\mathscr{F}} = \{X \in \mathscr{F}_C \mid \pi_1(X) \in [\![\delta]\!]^{\mathscr{F}} \ \&\ \pi_2(X) \in [\![\sigma]\!]^{\mathscr{F}}\}$
4. $[\![\sigma \to \kappa]\!]^{\mathscr{F}} = \{X \in \mathscr{F}_{\mathbb{S}D} \mid \forall Y \in [\![\sigma]\!]^{\mathscr{F}}.\, X \cdot Y \in [\![\kappa]\!]^{\mathscr{F}}\}$

where, for $X \in \mathscr{F}_C \cong (\mathscr{F}_D \times \mathscr{F}_S)_\perp$, $\pi_1(X) = \{\delta \in \mathscr{L}_D \mid \exists \sigma \in \mathscr{L}_S.\, \delta \times \sigma \in X\}$ and similarly for $\pi_2(X)$.

Proof. The first part is immediate from the definition of type interpretation and of filters. Concerning the remaining statements, we prove just (1) as the other ones are similar. First observe that $\uparrow \varphi \in [\![\varphi]\!]^{\mathscr{F}}$. Hence, if $X \in [\![\delta \to \tau]\!]^{\mathscr{F}}$ then $\delta \to \tau \in X$, which implies that $X \cdot \uparrow \delta = \uparrow \tau \in [\![\tau]\!]^{\mathscr{F}}$. Viceversa if $X \cdot Y \in [\![\tau]\!]^{\mathscr{F}}$ for all $Y \in [\![\delta]\!]^{\mathscr{F}}$ then in particular $X \cdot \uparrow \delta = \uparrow \tau \in [\![\tau]\!]^{\mathscr{F}}$, which implies that for some $\delta' \in \uparrow \delta$, $\delta' \to \tau \in X$; but then $\delta \leq_D \delta'$ so that $\delta' \to \tau \leq_D \delta \to \tau$ and therefore $\delta \to \tau \in X$ as X is upward closed. □

We can now build the type assignment system as the description, via type derivations, of the semantics of terms and types in the model $\mathscr{F}$. First, type contexts $\Gamma = \{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ are put into correspondence with environments $e_\Gamma : Var \to \mathscr{F}_D$ by setting $e_\Gamma(x_i) = \uparrow_D \delta_i$. Second, typing judgments translate the statements:

$$\Gamma \vdash V : \delta \iff ([\![V]\!]^{\mathscr{F}_D} e_\Gamma) \in [\![\delta]\!]^{\mathscr{F}} \iff \delta \in [\![V]\!]^{\mathscr{F}_D} e_\Gamma$$

and similarly, $\Gamma \vdash M : \tau \iff \tau \in [\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e_\Gamma$.

Third and final step, typing rules of the shape

$$\frac{\Gamma_1 \vdash P_1 : \varphi_1 \quad \cdots \quad \Gamma_n \vdash P_n : \varphi_n}{\Gamma \vdash Q : \psi}$$

are the translations of equations

$$[\![Q]\!]^{\mathscr{F}} e_\Gamma = \texttt{Filt}\{\psi \mid \varphi_1 \in [\![P_1]\!]^{\mathscr{F}} e_{\Gamma_1} \ \&\ \cdots \ \&\ \varphi_n \in [\![P_n]\!]^{\mathscr{F}} e_{\Gamma_n}\} \tag{10}$$

derived from Prop. 5.2 and Thr. 4.11. We take advantage of the factorization of the right hand sides of equations (10) into a set of types U and its closure $\texttt{Filt}(U)$, by adding the rules:

$$\frac{}{\Gamma \vdash P : \omega}\,(\omega) \qquad \frac{\Gamma \vdash P : \varphi \quad \Gamma \vdash P : \psi}{\Gamma \vdash P : \varphi \wedge \psi}\,(\wedge) \qquad \frac{\Gamma \vdash P : \varphi \quad \varphi \leq \psi}{\Gamma \vdash P : \psi}\,(\leq) \tag{11}$$

Therefore we are left to translate set inclusions

$$[\![Q]\!]^{\mathscr{F}} e_\Gamma \supseteq \{\psi \mid \varphi_1 \in [\![P_1]\!]^{\mathscr{F}} e_{\Gamma_1} \ \&\ \cdots \ \&\ \varphi_n \in [\![P_n]\!]^{\mathscr{F}} e_{\Gamma_n}\} \tag{12}$$

where ψ is neither an intersection nor an ω:

Definition 5.3. *(***Intersection type assignement system for λ_{imp}***) The system is obtained by adding to the rules (11) the following:*

$$\frac{}{\Gamma, x : \delta \vdash x : \delta}\,(var) \qquad \frac{\Gamma, x : \delta \vdash M : \tau}{\Gamma \vdash \lambda x.M : \delta \to \tau}\,(\lambda)$$

$$\frac{\Gamma \vdash V : \delta}{\Gamma \vdash [V] : \sigma \to \delta \times \sigma}\,(unit) \qquad \frac{\Gamma \vdash M : \sigma \to \delta' \times \sigma' \quad \Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''}{\Gamma \vdash M \star V : \sigma \to \delta'' \times \sigma''}\,(\star)$$

$$\frac{\Gamma, x : \delta \vdash M : \sigma \to \kappa}{\Gamma \vdash get_\ell(\lambda x.M) : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa}\,(get) \qquad \frac{\Gamma \vdash V : \delta \quad \Gamma \vdash M : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \quad \ell \notin dom(\sigma)}{\Gamma \vdash set_\ell(V, M) : \sigma \to \kappa}\,(set)$$

The rules in Definition 5.3 are obtained by instantiating the inclusion (12) to equations (6) - (9). In conclusion, we obtain for our type system the analogous of the "Type-semantics theorem" for intersection types and the ordinary λ-calculus (see [7], Thr. 16.2.7):

Theorem 5.4. *Let $e \models^{\mathscr{F}} \Gamma$ if and only if $e(x) \in [\![\delta]\!]^{\mathscr{F}}$ whenever $x : \delta \in \Gamma$; then*

1. $[\![V]\!]^{\mathscr{F}_D} e = \{\delta \in \mathscr{L}_D \mid \exists \Gamma.\, e \models^{\mathscr{F}} \Gamma \ \&\ \Gamma \vdash V : \delta\}$
2. $[\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e = \{\tau \in \mathscr{L}_{\mathbb{S}D} \mid \exists \Gamma.\, e \models^{\mathscr{F}} \Gamma \ \&\ \Gamma \vdash M : \tau\}$

Proof. Observe that $e \models^{\mathscr{F}} \Gamma$ if and only if $e_\Gamma(x) = \uparrow \delta \subseteq e(x)$ if $x : \delta \in \Gamma$; since any filter X is the union of the principal filters $\uparrow \delta$ with $\delta \in X$, the proof reduces to establishing that $[\![V]\!]^{\mathscr{F}_D} e_\Gamma = \{\delta \in \mathscr{L}_D \mid \Gamma \vdash V : \delta\}$, and similarly for $[\![M]\!]^{\mathscr{F}_{\mathbb{S}D}} e_\Gamma$; the latter are an easy induction over term interpretations into $\mathscr{F}$ for left to right inclusions, and over type derivations for the inclusions from right to left. □

In spite of the complexity of the construction we have been going through in the last sections, the payoff of our work is a system with just one typing rule for each syntactical construct in the grammar of the calculus, plus the "logical" rules (11) which are standard in intersection type systems with subtyping since [6].

Moreover, by the obvious interpretation $[\![\varphi]\!]^{\mathscr{D}}$ of types in a model $\mathscr{D}$, of which Prop. 5.2 is the instance in case of $\mathscr{F}$, we get soundness and completeness theorems for free. Indeed, without entering into the details, we claim that:

$$\Gamma \models V : \delta \iff \Gamma \vdash V : \delta \quad \text{and} \quad \Gamma \models M : \tau \iff \Gamma \vdash M : \tau.$$

where for a λ_{imp}-model $\mathscr{D}$ we write $\Gamma \models^{\mathscr{D}} T : \varphi$ if $[\![T]\!]^{\mathscr{D}} e \in [\![\varphi]\!]^{\mathscr{D}}$ for all $e \models^{\mathscr{D}} \Gamma$, and write $\Gamma \models T : \varphi$ if $\Gamma \models^{\mathscr{D}} T : \varphi$ for all $\mathscr{D}$.

6 Discussion and Related works

This work is a development of [14], where we considered a pure untyped computational λ-calculus, namely without operations nor constants. Therefore, the monad T and the respective unit and bind were generic, so that types cannot convey any specific information about the domain TD, nor about effects represented by the monad.

In the unpublished report [15] we sketched the construction of the type system for the state monad $\mathbb{S}$. This has been motivated on the ground of an operational interpretation of terms that we illustrated, culminating in the characterization of convergence via the type system. This is akin to the lazy λ-calculus [2], where convergent terms are characterized by the formula $(t \to t)_\perp$ of the endogenous logic of the calculus, in the sense of [1].

While deferring the exposition of the convergence characterization to a companion paper, here we have focussed on the construction of the type system itself. Since [6] we know that a λ-model can be constructed by taking the filters of types in an intersection type system with subtyping. The relation among the filter-model and Scott's D_∞ construction has been subject to extensive studies, starting with [12] and continuing with [16, 3, 4, 5]. In the meantime Abramsky's theory of domain logic in [1] provided a generalization of the same ideas to algebraic domains in the category of 2/3 SFP, of which ω-**ALG** is a (full) subcategory, based on Stone duality.

In the present work, we use the mathematical theories just mentioned, but reversing the process from semantics to types. Usually, one starts with a calculus and a type system, looking for a semantics and studying properties of the model. On the contrary, we move from a domain equation and the definition of the denotational semantics of the calculus of interest and synthesize a filter-model and an intersection type system. This synthetic use of domain logic is, in our view, prototypical w.r.t. the construction of type systems catching properties of any computational λ-calculus with operators. We expect that the study of such systems will be of help in understanding the operational semantics of such models, a topic that has been addressed in [13, 20], but with different mathematical tools.

A further research direction is to move from ω-**ALG** to other categories such as the category of relations. The latter is deeply related to non-idempotent intersection types [11, 10] that have been shown to catch intensional aspects of λ-calculi involving quantitative reasoning about computational resources. If the present approach can be rephrased in the category of relations, then our method could produce non-idempotent intersection type systems for effectful λ-calculi.

In the perspective of considering categories other than ω-**ALG**, it is natural to ask whether the synthesis of an "estrinisic" type system in the sense of Reynolds out of the denotational semantics of a calculus, either typed or not, can be described in categorical terms. Intersection type theories and related systems, as we have been using in this work, are a special case of refinement type systems; a starting point may be the framing of refinement type systems as functors in [21].

Acknowledgements. We are grateful to the anonymous referees for their comments and hints to categorical prospectives of the present work.

References

[1] S. Abramsky (1991): *Domain Theory in Logical Form*. Ann. Pure Appl. Log. 51(1-2), pp. 1–77, doi:10.1016/0168-0072(91)90065-T.

[2] S. Abramsky & C.-H.L. Ong (1993): *Full abstraction in the lazy lambda calculus*. Inf. Comput. 105(2), pp. 159–267, doi:10.1006/inco.1993.1044.

[3] F. Alessi, M. Dezani-Ciancaglini & F. Honsell (2004): *Inverse Limit Models as Filter Models*. In Delia Kesner, Femke van Raamsdonk & Joe Wells, editors: *HOR'04*, RWTH Aachen, Aachen, pp. 3–25.

[4] F. Alessi, F. Barbanera & M. Dezani-Ciancaglini (2006): *Intersection types and lambda models*. *Theor. Comput. Sci.* 355(2), pp. 108–126, doi:10.1016/j.tcs.2006.01.004.

[5] F. Alessi & P. Severi (2008): *Recursive Domain Equations of Filter Models*. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat & Mária Bieliková, editors: *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19-25, 2008, Proceedings*, *Lecture Notes in Computer Science* 4910, Springer, pp. 124–135, doi:10.1007/978-3-540-77566-9_11.

[6] H. P. Barendregt, M. Coppo & M. Dezani-Ciancaglini (1983): *A Filter Lambda Model and the Completeness of Type Assignment*. *J. Symb. Log.* 48(4), pp. 931–940, doi:10.2307/2273659.

[7] H. P. Barendregt, W. Dekkers & R. Statman (2013): *Lambda Calculus with Types*. Perspectives in logic, Cambridge University Press, doi:10.1017/CBO9781139032636.

[8] N. Benton, J. Hughes & E. Moggi (2002): *Monads and Effects*. In: *Applied Semantics, International Summer School, APPSEM 2000*, *Lecture Notes in Computer Science* 2395, Springer, pp. 42–122, doi:10.1007/3-540-45699-6_2.

[9] N. Benton, A. Kennedy, L. Beringer & M. Hofmann (2009): *Relational semantics for effect-based program transformations: higher-order store*. In António Porto & Francisco Javier López-Fraguas, editors: *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, ACM, pp. 301–312, doi:10.1145/1599410.1599447.

[10] A. Bucciarelli, T. Ehrhard & G. Manzonetto (2007): *Not Enough Points Is Enough*. In Jacques Duparc & Thomas A. Henzinger, editors: *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, *Lecture Notes in Computer Science* 4646, Springer, pp. 298–312, doi:10.1007/978-3-540-74915-8_24.

[11] D. de Carvalho (2018): *Execution time of λ-terms via denotational semantics and intersection types*. *Math. Struct. Comput. Sci.* 28(7), pp. 1169–1203, doi:10.1017/S0960129516000396.

[12] M. Coppo, M. Dezani-Ciancaglini, F. Honsell & G. Longo (1984): *Extended type structures and filter lambda models*. In G. Lolli, G. Longo & A. Marcja, editors: *Logic Colloquium 82*, North-Holland, Amsterdam, the Netherlands, pp. 241–262, doi:10.1016/S0049-237X(08)71819-6.

[13] U. Dal Lago, F. Gavazzo & P. B. Levy (2017): *Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method*. In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, IEEE Computer Society, pp. 1–12, doi:10.1109/LICS.2017.8005117.

[14] U. de'Liguoro & R. Treglia (2020): *The untyped computational λ-calculus and its intersection type discipline*. *Theor. Comput. Sci.* 846, pp. 141–159, doi:10.1016/j.tcs.2020.09.029.

[15] U. de'Liguoro & R. Treglia (2021): *Intersection Types for a Computational Lambda-Calculus with Global State, https://arxiv.org/abs/2104.01358*. arXiv:2104.01358.

[16] M. Dezani-Ciancaglini, F. Honsell & F. Alessi (2003): *A complete characterization of complete intersection-type preorders*. *ACM Trans. Comput. Log.* 4(1), pp. 120–147, doi:10.1145/601775.601780.

[17] F. Gavazzo (2019): *Coinductive Equivalences and Metrics for Higher-order Languages with Algebraic Effects*. Ph.D. thesis, University of Bologna, Italy. Available at http://amsdottorato.unibo.it/9075/.

[18] M. Hyland & J. Power (2006): *Discrete Lawvere theories and computational effects*. *Theor. Comput. Sci.* 366(1-2), pp. 144–162, doi:10.1016/j.tcs.2006.07.007.

[19] M. Hyland & J. Power (2007): *The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads*. *Electron. Notes Theor. Comput. Sci.* 172, pp. 437–458, doi:10.1016/j.entcs.2007.02.019.

[20] U. Dal Lago & F. Gavazzo (2019): *Effectful Normal Form Bisimulation*. In Luís Caires, editor: *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019*, *Lecture Notes in Computer Science* 11423, Springer, pp. 263–292, doi:10.1007/978-3-030-17184-1_10.

[21] P.-A. Melliès & N. Zeilberger (2015): *Functors are Type Refinement Systems*. In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*, ACM, pp. 3–16, doi:10.1145/2676726.2676970.

[22] E. Moggi (1991): *Notions of Computation and Monads*. *Inf. Comput.* 93(1), pp. 55–92, doi:10.1016/0890-5401(91)90052-4.

[23] G. D. Plotkin & J. Power (2002): *Notions of Computation Determine Monads*. In: *FOSSACS 2002*, *Lecture Notes in Computer Science* 2303, Springer, pp. 342–356, doi:10.1007/3-540-45931-6_24.

[24] G. D. Plotkin & J. Power (2003): *Algebraic Operations and Generic Effects*. *Appl. Categorical Struct.* 11(1), pp. 69–94, doi:10.1023/A:1023064908962.

[25] J. Power (2006): *Generic models for computational effects*. *Theor. Comput. Sci.* 364(2), pp. 254–269, doi:10.1016/j.tcs.2006.08.006.

[26] P. Wadler (1992): *The Essence of Functional Programming*. In: *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1992*, ACM Press, pp. 1–14, doi:10.1145/143165.143169.

[27] P. Wadler (1995): *Monads for Functional Programming*. In: *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques*, *Lecture Notes in Computer Science* 925, Springer, pp. 24–52, doi:10.1007/3-540-59451-5_2.