Theoretical Computer Science 973 (2023) 114082

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

From semantics to types: The case of the imperative λ-calculus ☆

Ugo de'Liguoro [a], Riccardo Treglia [b,*]

[a] Università di Torino, Turin, Italy
[b] Università di Bologna, Bologna, Italy

ARTICLE INFO

Article history:
Received 28 February 2022
Received in revised form 17 May 2023
Accepted 16 July 2023
Available online 22 July 2023

Keywords:
State monad
Imperative lambda calculus
Type assignment systems
Filter models

ABSTRACT

We study the logical semantics of an untyped λ-calculus equipped with operators representing read and write operations from and to a global store. Such a logic consists of an intersection type assignment system, which we derive from the denotational semantics of the calculus, based on the monadic approach to model computational λ-calculi.
The system is obtained by constructing a filter model in the category of ω-algebraic lattices, such that the typing rules can be recovered out of the term interpretation. By construction, the so-obtained type system satisfies the "type-semantics" property and completeness.

1. Introduction

The problem of integrating non-functional aspects into functional programming languages goes back to the early days of functional programming; nowadays, even procedural and object-oriented languages embody more and more features from declarative languages, renewing interest and motivations in the investigation of higher-order effectful computations.

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.

Side-effects and monads. Focusing on side-effects, the method used in the early studies to treat the store, e.g. [26,44,48] and [42], is essentially additive: update and dereferentiation primitives are added to an (often typed) λ-calculus, possibly with constructs to dynamically create and initialize new locations. Then, *a posteriori*, tools to reason about such calculi are built either by extending the type discipline or by means of denotational and operational semantics, or both.

The introduction of notions of computation as monads and of the computational λ-calculus by Moggi in [34] 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 [46,47], the introductory [13], and the large body of bibliography thereafter; a gentle introduction and further references can be found e.g. in [27]. The key idea is to model effectful computations as morphisms of the Kleisli category of a monad in [34], starting a very rich thread in the investi-

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.
E-mail addresses: ugo.deliguoro@unito.it (U. de'Liguoro), riccardo.treglia@unibo.it (R. Treglia).

https://doi.org/10.1016/j.tcs.2023.114082

gation of programming language foundations based on categorical semantics, which is still flourishing. The methodological advantage is that we have a uniform and abstract way of speaking of various kinds of effects, and the definition of equational logic to reason about them. At the same time computational effects, including side-effects, can be safely embodied into purely functional languages without disrupting their declarative nature, as shown by Wadler's [46,47] together with a long series of papers by the same author and others.

Monads alone 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* [38,39,37], Plotkin and Power have shown under which conditions effect operators live in the category of algebras of a computational monad, which is equivalent to the category of models of certain equational specifications, namely varieties in the sense of universal algebra [30].

The calculus λ_{imp} and its intersection type system. Initiating with [22], we have approached the issue of modeling effects in an untyped computational λ-calculus, studying its operational semantics by introducing a reduction relation and a syntactical convergence predicate characterized by an intersection type assignment system. Here we shall refer to such a calculus as the *computational core*, denoted $\lambda_{\copyright}$.

Intersection types are an extension of the Curry type assignment system which are a form of polymorphism in two senses. First, terms and types are distinct entities and the same term can have (infinitely) many types (see e.g. [31], Chap. 12); second intersection types embody a form of ad hoc polymorphism, where if M has both type σ and τ then $M : \sigma \wedge \tau$, and such a conjunction can be among semantically unrelated types.

The issue of using intersection types for $\lambda_{\copyright}$ is motivated by the ad hoc polymorphic type's ability to capture behavioral properties of terms such as weak and strong normalization and, simultaneously, bridge denotational and operational semantics via the concept of filter models. The fact that λ-models can be built by taking the set of filters of types generated by suitable subtyping relations substantiates the claim that intersection types are the *logical semantics* of the λ-calculus, with tremendous consequences in the study of the calculus itself and of its variants: see the recent survey [10]. Our aim is to extend the theory of intersection types and filter models to $\lambda_{\copyright}$ and to the computational λ-calculi in general.

The approach in [22] is limited to a pure calculus without constants, where the unit and bind operations are axiomatized by the monadic laws from Wadler's [47]. In [24] we add to the calculus syntax denumerably many operations get_ℓ and set_ℓ, indexed over an infinite set of locations, to access a global store. The resulting calculus, called λ_{imp}, is equipped with operational semantics and an intersection type assignment system which is shown to enjoy the fundamental properties of intersection types in the case of the ordinary λ-calculus; in particular, we show that the typings are invariant both by reduction and by expansion of the subject, and we characterize convergent terms by their typings.

The price we pay in [24] for such results resides in the complexity of the type assignment system, which involves four kinds of types and four subtyping relations defined by mutual induction. The main goal of the paper is to illustrate how a type assignment system can be derived out of the denotational semantics of a calculus; this is especially challenging in the case of λ_{imp} as well as for computational λ-calculi involving effects and algebraic operations.

From the denotational semantics to the type assignment system. Following Moggi [33], λ_{imp} can be modeled into a domain D satisfying the equation $D = D \to TD$, clearly reminiscent of Scott's $D = D \to D$ reflexive object, where T is instantiated to $\mathbb{S}$, a variant of the state monad in [34], called the partiality and state monad in [21,27]. The object part of the functor $\mathbb{S}$ is defined as $\mathbb{S}X = S \to (X \times S)_\perp$, where S is a suitable domain of states which we define as the (partial) stores over a domain D such that $D \cong D \to \mathbb{S}D$ in a suitable category of domains, for which we take ω-**Alg**, the category of ω-algebraic lattices and Scott continuous maps.

The actual choice of the monad $\mathbb{S}$ which determines the domain D above, although relevant to obtain the results in [24], is immaterial w.r.t. the subsequent steps in our construction, that can be easily adapted to other variants of the semantics of λ_{imp} and of similar calculi. Indeed, the method we follow would apply to any extension of $\lambda_{\copyright}$ and to variants of the ordinary untyped λ-calculus as well, provided they have semantics in ω-**Alg**.

The method is based on a fundamental result, originating from Scott information systems, that we recall in Theorem 4.4, namely the Representation Theorem, stating that any ω-algebraic lattice A is isomorphic to a domain of filters $\mathcal{F}_A$. Filters are subsets of $\mathcal{L}_A$, a language of intersection types whose syntax depends on A, that are upward closed and closed under finitary infs with respect to the pre-order $\leq_A$. The quotient of $(\mathcal{L}_A, \leq_A)$ is an inf-semilattice isomorphic to the set of compact points $K(A)$ of A, taken with the opposite of the ordering inherited from A. By algebraicity, such isomorphism extends to the isomorphism $\mathcal{F}_A \cong A$, so that we can describe A via the axiomatization of $\leq_A$, called *type theory* in [11], Chap. 13, which we denote Th_A. A key fact we use in our construction is that the definition of Th_A is functorial, as illustrated in [49] and used in [2], where Th_A is called the Lindenbaum algebra of A, namely its "logic". This implies that, for example, given $B, C \in \omega$-**Alg**, if $A = B \to C$ then we have a canonical method to produce $Th_{B\to C}$ from Th_B and Th_C such that $\mathcal{F}_{B\to C} \cong \mathcal{F}_B \to \mathcal{F}_C \cong B \to C$.

At this point, the problem is to describe $D \cong D \to \mathbb{S}D$ in terms of the domains of filters $\mathcal{F}_D$ and $\mathcal{F}_{\mathbb{S}D}$, that is of the respective type theories. Not surprisingly, the definition of the theories Th_D and $Th_{\mathbb{S}D}$ depend on each other as a consequence of the fact that D is the model of a recursive type. One way to come out of the difficulty would be to parallel the construction of these theories to that of the domains $D_{n+1} = D_n \to \mathbb{S}D_n$ in the inverse limit construction of D, along the lines of the description of D_∞ models of the λ-calculus as filter models (see section 16.3 of [11]). We avoid such a daunting task by the same way implicit in [9], which is essentially grounded on the solution of domain equations

in the category of information systems and approximable mappings in [49]. Indeed, intersection types equipped with the subtyping relation can be seen as a particular kind of information system [16]. The construction consists of defining both $\mathcal{L}_D$ and Th_D on the one hand, and $\mathcal{L}_{\mathbb{S}D}$ and $Th_{\mathbb{S}D}$ on the other, simply by mutual induction, and then proving the isomorphism $\mathcal{F}_D \cong \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$ via the fact that $\mathcal{F}_D$ coincides with $\mathcal{F}_{D\to\mathbb{S}D}$.

Eventually, we come to the payoff of this long journey through the theory of intersection types and filter models. The key result in [9] is that the interpretation of an ordinary λ-term $[\![M]\!]$ (where M is closed for simplicity) in the model $\mathcal{F}_D \cong D$ coincides with the set of types σ such that $\vdash M : \sigma$ is derivable in the system they were studying. This is called the "type-semantics theorem" in [11], Theorem 16.2.7, which we establish here as Theorem 6.3 in the case of λ_{imp}. Among the pleasant consequences of this theorem, we have the completeness of the type assignment system and, in general, the warranty that relevant properties of the calculus can be formally established via a logical system.

Now, at the heart of such a result is the fact that there is a close correspondence between the interpretation of the operators involved in the definition of the semantics of terms into the filter model and the typing rules in the type assignment system, and so we reverse the approach: instead of discovering such a correspondence *a posteriori*, we use the interpretation of the operators, that are defined in general for any model of λ_{imp}, in the case of the filter model, getting the typing rules from the semantics of the terms in this particular model. Thus, the type-semantics theorem as well as the completeness of the type assignment system w.r.t. the interpretation of types as the set of filters to which they belong, come out for free.

Summary and results. In Section 2 we outline the syntax of the untyped imperative λ-calculus λ_{imp}. 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 modeling algebraic operators over the monad $\mathbb{S}$ and formalizing the model of λ_{imp}. In Section 4, which is the central part of the paper, we construct a filter model of λ_{imp}.

Section 5 explains how to derive the type assignment system from the filter model construction. We conclude with Section 6 establishing the type semantics and completeness theorems for our type system. Finally, Section 7 is devoted to the discussion of our results and to related works.

We assume familiarity with λ-calculus, intersection types, and domain theory; comprehensive references are Barendregt's book [11], Part III and [3]. Notions from domain theory, computational monads, and algebraic effects, are shorty recalled in the paper; for further references see e.g. [5], [3], [13], and [27] Chap. 3.

The present paper is a revised and extended version of [23].

2. An untyped imperative λ-calculus

Imperative extensions of the λ-calculus, both typed and type-free, are usually based on the call-by-value λ-calculus, 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 Moggi [34], we consider the computational core $\lambda_{\copyright}$, to which we add syntax denoting algebraic effect operations à la Plotkin and Power [38,39,37] over a suitable state monad.

Let $\mathbf{L} = \{\ell_0, \ell_1, \ldots\}$ be a denumerable set of abstract *locations*. Borrowing notation from [27], Chap. 3, we consider denumerably many operator symbols get_ℓ and set_ℓ, obtaining:

Definition 2.1 *(Term syntax).*

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

As for $\lambda_{\copyright}$, terms are of either sorts *Val* or *Com*, representing values and computations, respectively. The new constructs are $get_\ell(\lambda x.M)$ and $set_\ell(V, M)$. 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.

With respect to the syntax of the "imperative λ-calculus" in [27], we do not have the *let* construct, nor the application VW among values. The justification is the same as for the computational core. These constructs are definable:

$$\mathsf{let}\, x := M \,\mathsf{in}\, N \;\equiv\; M \star (\lambda x.N) \qquad VW \;\equiv\; [W] \star V$$

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:

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

representing location dereferentiation and assignment. 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. In fact, 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. This will be, of course, a major challenge when designing the type assignment system in the next sections.

3. Denotational semantics

We illustrate the denotational semantics of the calculus λ_{imp} introduced in Section 2 in the category of domains. The carrier of the model is a 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, while the interpretation depends on certain algebraic operations associated to the monad $\mathbb{S}\,X = S \to (X \times S)_\perp$ which is a variant of the state monad in Moggi [34], called the partiality and state monad in [21].

We now recall the definition of monad and algebraic operation over a monad, and fix notation.

The partiality and state monad. We recall Wadler's type-theoretic definition of monads [46,47], which is 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, $\mathcal{C}$ will be a *concrete ccc*, namely a ccc that is a concrete category, such that the full and faithful functor from $\mathcal{C}$ to **Set** is well-behaved with respect to products and exponents; this implies that it $\mathcal{C}$ is well-pointed (such a category is called *strictly concrete* in [8], Definition 5.5.8, and when it has a reflexive object it is a model of the untyped λ-calculus), and can be seen as a category of sets and functions. Examples that are relevant to us are the category of Scott domains with continuous functions and its full subcategory ω-**Alg** of algebraic lattices with a countable basis. It is known that ω-**Alg** is closed under lifting and that it is Cartesian closed (see e.g. [5], Cor. 4.1.6), which suffices for the following definitions to make sense of it, and for Equation (2) to have a solution. More details about algebraicity are recalled in the Section 4, where they are of use.

Definition 3.1. (Computational Monad [47] §3) Let $\mathcal{C}$ be a concrete ccc. A *functional computational monad*, henceforth just *monad* over $\mathcal{C}$ is a triple $(T, unit, \star)$ where T is a map over the objects of $\mathcal{C}$, 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:

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

where $\lambda\!\!\lambda d.(f\ d \star g)$ is the lambda notation for $d \mapsto f\ d \star g$.

Remark 3.2. The T in Definition 3.1 is a *strong monad* over $\mathcal{C}$. Indeed it is a *Kleisli triple* $(T, \eta, \_^\dagger)$ (see [34], Definition 1.2), where

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

The first equality, involving the η, called the *unit* of the triple, is just a notational variation of the coercion of values into computations. The latter, instead, establishes the connection between the bind 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 $f = f^\dagger \circ \eta_X$.

The map T turns out to be a functor whose morphism action for $h : X \to Y$ is $Th = (unit_Y \circ h)^\dagger$, where $(unit_Y \circ h)^\dagger a = a \star \lambda\!\!\lambda x.\, unit_Y(h\,x)$, and indeed (T, η, μ) is a *monad* (see [34], Definition 1.5), with *multiplication* $\mu : T^2 \to T$ having components $\mu_X = \mathsf{id}^\dagger_{TX} = \lambda\!\!\lambda z.\, z \star \mathsf{id}_{TX}$.

Finally the *tensorial strength* ([34], Definition 3.2) is a natural transformation with components $t_{X,Y} : X \times TY \to T(X \times Y)$ defined by

$$t_{X,Y}(x, z) = z \star \lambda\!\!\lambda y.\, unit_{X\times Y}(x, y)$$

that are well-behaved w.r.t. the unit and the multiplication. Since $\mathcal{C}$ is a concrete ccc, it has enough points, hence the tensorial strength t is unique ([34], Proposition 3.4).

Now, let us look closer at how monads model effectful computations. Let X be a domain of values; then TX is the domain, or the type, of computations with values in X. In general, TX has a richer structure than X, modeling partial computations, exceptions, non-determinism, etcetera, and 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. 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 $f(x) \in TY$.

We now introduce the *partiality and state monad* which is a variant of Moggi's state monad adapted to the case of partial computations.

Given a domain X, let $X_\perp$ be the *lifting* of X, namely the poset $X \cup \{\perp\}$ (with $\perp \notin X$) where $x \sqsubseteq_{X_\perp} x'$ if either $x = \perp$ or $x \sqsubseteq_X x'$.

Definition 3.3. (Partiality and state monad) Given a domain S 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:

$$unit\ x ::= \lambda\varsigma.(x, \varsigma)$$
$$(c \star f)\,\varsigma = f^\dagger(c)(\varsigma) ::= \begin{cases} f\,(x)(\varsigma') & \text{if } c(\varsigma) = (x, \varsigma') \neq \perp \\ \perp & \text{if } c(\varsigma) = \perp \end{cases}$$

where we omit subscripts.

In the following, we shall abbreviate the definition of $c \star f$ by

$$(c \star f)\,\varsigma = \mathsf{let}\,(x, \varsigma') := c\,(\varsigma)\,\mathsf{in}\,f\,(x)(\varsigma')$$

which is strict in $c(\varsigma)$. Then, if $h : X \to Y$ we compute $(\mathbb{S}\,h) = (unit_Y \circ h)^\dagger$ by

$$\begin{aligned} (\mathbb{S}\,h)\,c\,\varsigma &= \mathsf{let}\,(x, \varsigma') := c(\varsigma)\,\mathsf{in}\,(unit_Y \circ h)\,x\,\varsigma' \\ &= \mathsf{let}\,(x, \varsigma') := c(\varsigma)\,\mathsf{in}\,(h(x), \varsigma') \end{aligned}$$

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 state 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.

To relate Equation (1) to the monad $\mathbb{S}$ we have to be more precise about the domain S of states.

Definition 3.5 *(Partial stores and the monads $\mathbb{S}_X$).* Given a domain X of values we define the domain of *partial stores* (shortly *stores*) over X as $FX = \Pi_{\mathbf{L}}(X_\perp)$, namely the denumerable product of $X_\perp$ with itself indexed over $\mathbf{L}$.

We define $\mathbb{S}_X$ as the partiality and state monad $\mathbb{S}$ where the state domain S is FX, namely the domain of stores over X.

The above definition is rather unusual in denotational semantics, where it is common to think of stores as partial maps from locations to values. For this reason and by abusing notation, we shall write $FX = (X_\perp)^{\mathbf{L}}$ instead of $\Pi_{\mathbf{L}}(X_\perp)$ and $\varsigma(\ell)$ for $\pi_\ell(\varsigma)$. Strictly speaking, the set $(X_\perp)^{\mathbf{L}}$ is exponentiation in **Set**, not in ω-**Alg** since $\mathbf{L}$ is a set and not a lattice; however, by taking the pointwise ordering noted $\varsigma \sqsubseteq_{(X_\perp)^{\mathbf{L}}} \varsigma'$, if $\varsigma(\ell) \sqsubseteq_{X_\perp} \varsigma'(\ell)$ for all $\ell \in \mathbf{L}$, we have that $(X_\perp)^{\mathbf{L}}$ is a lattice that is isomorphic to $\Pi_{\mathbf{L}}(X_\perp)$, and it is algebraic if X is such. Finally, $\varsigma(\ell) = \perp$ represents the fact that $\ell \notin dom(\varsigma)$, which will be important in our construction.

Both seeing FX as an infinite product or as a space of (partial) functions, the definition of F is functorial; in particular, we may set $Fg = g_\perp \circ \_ : (X_\perp)^{\mathbf{L}} \to (Y_\perp)^{\mathbf{L}}$, where $g : X \to Y$ and $g_\perp(x) = g(x)$ if $x \in X$, and $g_\perp(\perp) = \perp$, otherwise.

It turns out that F is locally continuous, following arguments in [41], the same is true of all the functors involved in Equation (1). This implies that the inverse limit technique can be used to solve this equation.

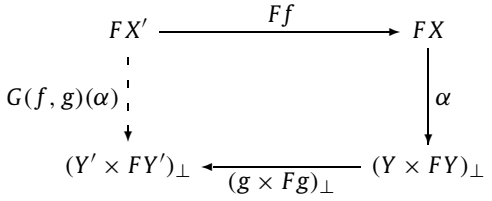
A domain equation. Turning to Equation (1), it is tempting to set $S = FX$ to solve the domain equation by means of standard techniques. However, the domain $FX = (X_\perp)^{\mathbf{L}}$ depends on X itself; in contrast, for defining $\mathbb{S}\,X$ the domain S has to be

fixed, since otherwise, the definition of the bind 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 $G : \omega\text{-}\mathbf{Alg}^{op} \times \omega\text{-}\mathbf{Alg} \to \omega\text{-}\mathbf{Alg}$ by

$$G(X, Y) = FX \to (Y \times FY)_\perp$$

whose action on morphisms is illustrated by the diagram:



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 in ω-**Alg** the initial solution to the domain equation (which is the same as Equation (1)):

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

Theorem 3.6. *There exists a domain D such that the state monad $\mathbb{S}_D$ with state domain $S = (D_\perp)^{\mathbf{L}}$ is a solution in ω-**Alg** to the domain equation:*

$$D = D \to \mathbb{S}_D\, D$$

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

Proof. Take D to be the (initial) solution to Equation (2); now if $S = FD = (D_\perp)^{\mathbf{L}}$ 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 $f : X \to TX$ as the meaning of a program with effects does not tell anything about effects themselves, which in the case of the state monad 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 [38,39,37] a theory of *algebraic operations*. A gentle introduction to the theory can be found in [27], Chap. 3, from which we borrow the notation.

Suppose that T is a monad over a category $\mathcal{C}$ with terminal object **1** and all finite products; then an algebraic operation **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

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

where $f : X \to TY$ and $\Pi_n f^\dagger = f^\dagger \times \cdots \times f^\dagger : (TX)^n \to (TY)^n$. In case of a concrete ccc, **1** is a singleton and products are sets of tuples, ordered componentwise. Then $\mathbf{op}_X$ is an operation of arity n of an algebra with carrier TX; since $f^\dagger : TX \to TY$, 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:

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

namely the functions $f^\dagger$ are homomorphisms w.r.t. **op**.

Algebraic operations $\mathbf{op}_X : (TX)^n \to TX$ do suffice in case T is, say, the non-determinism 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 [39] (and further studied in [29]), which are certain natural transformations satisfying a coherence condition. Below we instantiate such a construction, which is carried out in a suitable enriched category, in the case of concrete ccc's.

Definition 3.7 *(Algebraic operation).* An *algebraic operation* **op** over the computational monad T with *parameter object* P and *arity object* A is a family of morphisms

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

satisfying the coherence condition that, for all $f : X \to TY$ the following diagram commutes:

$$\begin{array}{ccccc} P \times (TX)^A & \xrightarrow{\mathbf{op}_X} & TX & & X \\ \downarrow{\scriptstyle \mathsf{id}_P \times (f^\dagger \circ -)} & & \downarrow{\scriptstyle f^\dagger} & & \downarrow{\scriptstyle f} \\ P \times (TY)^A & \xrightarrow[\mathbf{op}_Y]{} & TY & & TY \end{array}$$

namely, the following equation holds:

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

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

Proposition 3.8. *Any algebraic operation* ***op*** *is a natural transformation.*

Proof. We have to show that for every $h : X \to Y$ the following diagram commutes:

$$\begin{array}{ccccc} P \times (TX)^A & \xrightarrow{\mathbf{op}_X} & TX & & X \\ \downarrow{\scriptstyle \mathsf{id}_P \times (Th)^A} & & \downarrow{\scriptstyle Th} & & \downarrow{\scriptstyle h} \\ P \times (TY)^A & \xrightarrow[\mathbf{op}_Y]{} & TY & & Y \end{array}$$

where $(Th)^A : (TX)^A \to (TY)^A$ is $(Th)^A\, u = Th \circ u$ for $u \in (TX)^A$ (identifying $(TX)^A$ and $(TY)^A$ with the respective hom-sets). The commutativity of the above diagram is expressed by the equation

$$Th \circ \mathbf{op}_X = \mathbf{op}_Y \circ (\mathsf{id}_P \times (Th)^A) = \mathbf{op}_Y \circ (\mathsf{id}_P \times (Th \circ -))$$

namely for $p \in P$:

$$(Th)(\mathbf{op}_X(p,u)) = \mathbf{op}_Y(p, Th \circ u) \tag{5}$$

Now Equation (4) implies Equation (5) since $Th = (unit_Y \circ h)^\dagger$ and $(Th)^A = Th \circ -$. □

Denotational semantics of terms. We begin by defining the interpretations of the constants get_ℓ and set_ℓ as algebraic operations over $\mathbb{S}_D$, where the intended choice of D is the domain constructed in Theorem 3.6, but it is just a parameter as far as the next definition is concerned.

Definition 3.9. Let D be a domain and $\mathbb{S}_D$ the state monad with stores over D. Then we define the families of functions $\mathsf{get}_{\ell,X}$ and $\mathsf{set}_{\ell,X}$ indexed over the objects X of ω-**Alg** by:

1. $\mathsf{get}_{\ell,X} : \mathbf{1} \times (\mathbb{S}_D X)^D \to \mathbb{S}_D X \cong (\mathbb{S}_D X)^D \to \mathbb{S}_D X$ and

$$\mathsf{get}_{\ell,X}(u)(\varsigma) \;=\; \mathsf{let}\, d := \varsigma(\ell)\, \mathsf{in}\, u\, d\, \varsigma$$

   which is strict in $\varsigma(\ell)$.
2. $\mathsf{set}_{\ell,X} : D \times (\mathbb{S}_D X)^{\mathbf{1}} \to \mathbb{S}_D X \cong D \times (\mathbb{S}_D X) \to \mathbb{S}_D X$ and

$$\mathsf{set}_{\ell,X}(d,c)(\varsigma) \;=\; c\,(\varsigma[\ell \mapsto d])$$

for all $\varsigma \in S = (D_\perp)^{\mathbf{L}}$, where $c \in \mathbb{S}_D X = S \to (D \times S)_\perp$ and $\varsigma[\ell \mapsto d]$ is the store which sends ℓ to d and it is equal to ς otherwise.

The functions $\mathsf{get}_{\ell,X}$ and $\mathsf{set}_{\ell,X}$ correspond to $lookup_X$ and $update_X$ in [39], respectively, up to an irrelevant permutation of the arguments and the fact that we do not consider **L** as an object of ω-**Alg**; not surprisingly they are the components of two algebraic operations over $\mathbb{S}_D$, where $P = \mathbf{1}$ and $A = D$ in case of $\mathsf{get}_{\ell,X}$, and $P = D$ and $A = \mathbf{1}$ in case of $\mathsf{set}_{\ell,X}$ which is established in the next lemma and its corollary.

Lemma 3.10. *Let $h : X \to Y$ and $f : X \to \mathbb{S}_D X$, then:*

1. $get_{\ell,X}(u) \star f = get_{\ell,X}(\lambda x.\,(u\, x \star f))$

2. $set_{\ell,X}(d,c) \star f = set_{\ell,X}(d, c \star f)$

Proof. In the following let $\varsigma \in S$ be arbitrary; then

Part (1):

$$\begin{aligned}
(\mathsf{get}_{\ell,X}(u) \star f)\,\varsigma &= \mathsf{let}\,(x,\varsigma') := \mathsf{get}_{\ell,X}(u)(\varsigma)\,\mathsf{in}\,f\,x\,\varsigma' \\
&= \mathsf{let}\,(x,\varsigma') := (\mathsf{let}\,d := \varsigma(\ell)\,\mathsf{in}\,u\,d\,\varsigma)\,\mathsf{in}\,f\,x\,\varsigma' \\
&= \mathsf{let}\,d := \varsigma(\ell)\,\mathsf{in}\,(\mathsf{let}\,(x,\varsigma') := u\,d\,\varsigma\,\mathsf{in}\,f\,x\,\varsigma') \quad (*) \\
&= \mathsf{let}\,d := \varsigma(\ell)\,\mathsf{in}\,(u\,d \star f)\,\varsigma \\
&= \mathsf{let}\,d := \varsigma(\ell)\,\mathsf{in}\,(\lambda x.\,(u\,x \star f))\,d\,\varsigma \\
&= (\mathsf{get}_{\ell,X}(\lambda x.\,(u\,x \star f)))\,\varsigma
\end{aligned}$$

where $(*)$ follows by the associativity property of the let-expressions:

$$\mathsf{let}\,x := (\mathsf{let}\,y := e\,\mathsf{in}\,e')\,\mathsf{in}\,e'' = \mathsf{let}\,y := e\,\mathsf{in}\,(\mathsf{let}\,x := e'\,\mathsf{in}\,e'')$$

Indeed, if either $\varsigma(\ell)$ or $u\,(\varsigma(\ell))\,\varsigma$ are undefined (namely equal to $\bot$), then both $\mathsf{let}\,(x,\varsigma') := (\mathsf{let}\,d := \varsigma(\ell)\,\mathsf{in}\,u\,d\,\varsigma)\,\mathsf{in}\,f\,x\,\varsigma'$ and $\mathsf{let}\,d := \varsigma(\ell)\,\mathsf{in}\,(\mathsf{let}\,(x,\varsigma') := u\,d\,\varsigma\,\mathsf{in}\,f\,x\,\varsigma')$ are undefined. Otherwise, these expressions are both equal to $f\,\pi_1(u\,(\varsigma(\ell))\,\varsigma)\,\pi_2(u\,(\varsigma(\ell))\,\varsigma)$.

Part (2):

$$\begin{aligned}
(\mathsf{set}_{\ell,X}(d,c) \star f)\,\varsigma &= \mathsf{let}\,(x,\varsigma') := \mathsf{set}_{\ell,X}(d,c)(\varsigma)\,\mathsf{in}\,f\,x\,\varsigma' \\
&= \mathsf{let}\,(x,\varsigma') := c\,(\varsigma[\ell \mapsto d])\,\mathsf{in}\,f\,x\,\varsigma' \\
&= (c \star f)(\varsigma[\ell \mapsto d]) \\
&= \mathsf{set}_{\ell,X}(d, c \star f)(\varsigma) \quad \square
\end{aligned}$$

Corollary 3.11. *For all $\ell \in \mathbf{L}$ the functions $get_{\ell,X}$ and $set_{\ell,X}$ are the components at X of two algebraic operations get_ℓ and set_ℓ over $\mathbb{S}_D$.*

Proof. Part (1) of Lemma 3.10 is an instance of Equation (4), where $\mathsf{get}_{\ell,X}(u) \star f$ is $\mathsf{get}_{\ell,X}(*,u) \star f$ with $*$ as the unique element of $\mathbf{1}$.

In part (2) of the lemma, $c \in \mathbb{S}_D X$ is identified with $\lambda\_.c \in (\mathbb{S}_D X)^{\mathbf{1}}$, since $A = \mathbf{1}$ in case of set_ℓ and $\lambda\_.c \in (\mathbb{S}_D X)^{\mathbf{1}}$ is the function picking $c \in \mathbb{S}_D X$ (a point of $\mathbb{S}_D X$ in categorical terms). Hence, up to the isomorphism $\mathbb{S}_D X \cong (\mathbb{S}_D X)^{\mathbf{1}}$, we have e.g.

$$\begin{aligned}
\mathsf{set}_{\ell,X}(d, \lambda_.c) \star f &= \mathsf{set}_{\ell,X}(d,c) \star f \\
&= \mathsf{set}_{\ell,X}(d, c \star f) && \text{by Lemma 3.10.2} \\
&= \mathsf{set}_{\ell,X}(d, \lambda_.(c \star f)) \\
&= \mathsf{set}_{\ell,X}(d, \lambda x.((\lambda_.c)(x) \star f))
\end{aligned}$$

which is an instance of Equation (4). $\square$

We are now in place to define what is a model of λ_{imp}. Let $Env_D = Var \to D$ be the set of environments interpreting term variables into D, then:

Definition 3.12. A *λ_{imp}-model* is a structure $\mathcal{D} = (D, [\![\cdot]\!]^D, [\![\cdot]\!]^{\mathbb{S}D})$ where D, $[\![\cdot]\!]^D : Val \to Env_D \to D$ and $[\![\cdot]\!]^{\mathbb{S}D} : Com \to Env_D \to \mathbb{S}_D D$ are such that:

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

$$\begin{aligned}
[\![x]\!]^D e &= e(x) \\
[\![\lambda x.M]\!]^D e &= \Psi(\lambda z \in D.\,[\![M]\!]^{\mathbb{S}D} e[x \mapsto z]) \\
[\![\,[V]\,]\!]^{\mathbb{S}D} e &= unit\,([\![V]\!]^D e) \\
[\![M \star V]\!]^{\mathbb{S}D} e &= ([\![M]\!]^{\mathbb{S}D} e) \star \Phi([\![V]\!]^D e) \\
[\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e &= \mathsf{get}_{\ell,D}(\Phi([\![\lambda x.M]\!]^D e)) \\
[\![set_\ell(V,M)]\!]^{\mathbb{S}D} e &= \mathsf{set}_{\ell,D}([\![V]\!]^D e, [\![M]\!]^{\mathbb{S}D} e)
\end{aligned}$$

By unraveling definitions and applying their left and right-hand sides to an arbitrary store $\varsigma \in S$, the last two clauses can be written:

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

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

Proposition 3.13. *The following equations are true in $\mathcal{D}$:*

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 3.12 and straightforward calculations. The only interesting cases are (4) and (5).

In case (4), let $e' = e[x \mapsto z]$ where z is intended to vary over D, then by omitting the apices and the isomorphism Φ:

$$\begin{aligned} [\![get_\ell(\lambda x.M) \star W]\!] e &= \mathsf{get}_{\ell,D}(\dot{\lambda} z.[\![M]\!]e') \star [\![W]\!]e \\ &= \mathsf{get}_{\ell,D}(\dot{\lambda} z.([\![M]\!]e' \star [\![W]\!]e)) \quad \text{by Lemma 3.10.1} \\ &= \mathsf{get}_{\ell,D}(\dot{\lambda} z.([\![M]\!]e' \star [\![W]\!]e')) \quad (*) \\ &= \mathsf{get}_{\ell,D}(\dot{\lambda} z.[\![M \star W]\!]e') \\ &= [\![get_\ell(\lambda x.(M \star W))]\!] \end{aligned}$$

where in step (*) we use the fact that $[\![W]\!]e = [\![W]\!]e'$ because $x \notin FV(W)$, using Definition 3.12 in the remaining steps.

The case (5) is a direct consequence of Lemma 3.10.2. □

4. The filter model construction

This section is the central one in the paper, where we construct a filter model of λ_{imp} which is isomorphic to the domain found in Theorem 3.6, but it is more informative since it provides a concrete description of such a model in terms of its "logic", in the sense of [2]. By instantiating the definition of term interpretation from Definition 3.12 in the case of the filter model, we obtain a characterization of the types belonging to the interpretation of terms in Theorem 4.20, from which we recover the rules of the type assignment system for λ_{imp} in the subsequent section.

We begin with some basic facts from domain theory. Recall that a non-empty subset $X \subseteq D$ of a partial order $(D, \sqsubseteq_D)$ is *directed* if for all $x, y \in X$ there exists $z \in X$ with $x \sqsubseteq_D z \sqsupseteq_D y$. D is said to have all *directed sups* if the sup $\bigsqcup X \in D$ exists whenever X is directed. We often write $\bigsqcup^{\uparrow} X$ to stress that $\bigsqcup X$ is a directed sup. A point $e \in D$ is *compact* if for all directed X, $e \sqsubseteq_D \bigsqcup^{\uparrow} X$ implies $e \sqsubseteq_D x$ for some $x \in X$; the set of compact points of D is denoted $K(D)$.

The category ω-**Alg** is the full subcategory of the category of domains whose objects are complete lattices $(D, \sqsubseteq_D)$ such that $K(D)$ is countable and D is *algebraic*, namely $d = \bigsqcup^{\uparrow}\{e \in K(D) \mid e \sqsubseteq_D d\}$ is a directed sup for all $d \in D$. In case of algebraic domains the upward cones $\uparrow d = \{e \mid d \sqsubseteq e\}$ of compact points d form a basis for the Scott topology over D; therefore abusing terminology the set $K(D)$ is often referred to as a "basis" for such a topology.

As a complete lattice, D has arbitrary sups, but a morphism $f : D \to E \in \omega$-**Alg** is just a *Scott-continuous* map preserving directed sups, that is $f(\bigsqcup^{\uparrow} X) = \bigsqcup^{\uparrow}_{x \in X} f(x)$; in general, a morphism of ω-**Alg** does not necessarily preserve the sup $\bigsqcup Y$ for arbitrary $Y \subseteq D$.

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

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

In the literature, the operator $\wedge$ is called *intersection*, and ω_A the *universal type*. The theory Th_A is called *intersection type theory with universe* in [11] Definition 13.1.4, where ω_A is written $\mathtt{U}$. We write $\alpha =_A \beta$ if $\alpha \leq_A \beta \leq_A \alpha$, which is a congruence

w.r.t. $\leq_A$. By this the quotient $\mathcal{L}_A/_{\leq_A}$ is an inf-semilattice, with (the extension of) $\wedge$ as inf and (the equivalence class of) ω_A as top element.

A natural model of Th_A is the powerset of some set A. Any type $\alpha \in \mathcal{L}_A$ is interpreted as a subset $[\![\alpha]\!] \subseteq A$ such that $\alpha \leq_A \beta$ if and only if $[\![\alpha]\!] \subseteq [\![\beta]\!]$; hence $[\![\alpha \wedge \beta]\!] = [\![\alpha]\!] \cap [\![\beta]\!]$ and $[\![\omega_A]\!] = A$.

Informally, we identify Th_A with the set of inequalities $\alpha \leq_A \beta$, so that, abusing terminology, we shall also say that $\leq_A$ is a type theory. Also, we reason on types up to $=_A$ and treat $\wedge$ as commutative, associative and idempotent; consequently we use the notations $\bigwedge_{i=1}^n \alpha_i$ and $\bigwedge_{i\in I} \alpha_i$ for $\alpha_1 \wedge \cdots \wedge \alpha_n$ where $I = \{1, \ldots, n\}$ and assume that all the α_i are distinct; in particular $\bigwedge_{i\in\emptyset} \alpha_i = \omega_A$.

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

For any $X \subseteq \mathcal{L}_A$ we write $\uparrow_A X$ for the least filter in $\mathcal{F}_A$ including X; this can be equivalently defined as $\bigcap\{F \in \mathcal{F}_A \mid X \subseteq F\}$ or as the set $\{\beta \in \mathcal{L}_A \mid \exists n, \alpha_1, \ldots, \alpha_n \in X.\ \bigwedge_{i=1}^n \alpha_i \leq_A \beta\}$; in particular the principal filter over α is $\uparrow_A\{\alpha\}$, also written $\uparrow_A \alpha$, coinciding with the upward closure of $\{\alpha\}$. The mapping $\uparrow\cdot$ is evidently a closure operator; in the following we shall write just $\uparrow X$ and $\uparrow\alpha$ whenever A is understood.

In domain theory, the notion of a filter is just dual to that of an *ideal*, which is a downward closed and directed subset of a poset. In the case of algebraic lattices, the set $K(D)$ with the ordering inherited from D is a sup-semilattice, and it can be shown that the set $\mathcal{I}_{K(D)}$ of ideals over $K(D)$ taken with the subset ordering is isomorphic to D. $\mathcal{I}_{K(D)}$ is called the *ideal completion* of $K(D)$.

Dually the poset $K^{op}(D)$, which is $K(D)$ with the opposite ordering, is an inf-semilattice, so that $(\mathcal{F}_{K^{op}(D)}, \subseteq)$, called the *filter completion* of $K^{op}(D)$, is again isomorphic to D, hence $\mathcal{I}_{K(D)} \cong D \cong \mathcal{F}_{K^{op}(D)}$. In the following we write just $\mathcal{F}_D$ for $\mathcal{F}_{K^{op}(D)}$.

Below we exploit that Th_A is the axiomatization of an inf-semilattice to construct a domain $\mathcal{F}_A$ by filter completion of the inf-semilattice of types induced (after a quotient) by the preorder $\leq_A$: see Theorem 4.4. This theorem is the fundamental fact about intersection types and ω-algebraic lattices; before its proof, we recap some basic properties of the poset $(\mathcal{F}_A, \subseteq)$ in the following lemma.

Lemma 4.3. *Consider the poset $(\mathcal{F}_A, \subseteq)$ and let $\mathcal{X} \subseteq \mathcal{F}_A$ be a family of filters:*

1. *the sets $\bigcap \mathcal{X}$ and $\uparrow(\bigcup \mathcal{X})$ are the inf and the sup of $\mathcal{X}$ in $\mathcal{F}_A$, respectively;*
2. *if $\mathcal{X}$ is directed w.r.t. $\subseteq$, then $\uparrow(\bigcup \mathcal{X}) = \bigcup \mathcal{X}$.*

Proof. Part (1) is immediate; in particular, that $\bigcap \mathcal{X}$ is a filter follows by the fact that any $F \in \mathcal{X}$ is such, and the intersection of filters must satisfy the same closure properties of all the elements of $\mathcal{X}$. Note that the closure $\uparrow\cdot$ is necessary in $\uparrow(\bigcup \mathcal{X})$: take $\mathcal{X} = \{\uparrow\alpha, \uparrow\beta\}$, with types $\alpha, \beta \in \mathcal{L}_A$ unrelated w.r.t. $\leq_A$, then $\alpha \wedge \beta \notin \uparrow\alpha \cup \uparrow\beta$.

Part (2): any directed $\mathcal{X}$ is non empty, hence there is an $F \in \mathcal{X}$ such that $\omega_A \in F \subseteq \bigcup \mathcal{X}$, hence $\bigcup \mathcal{X}$ is nonempty. If $\alpha \in \bigcup \mathcal{X}$ then there is an F such that $\alpha \in F \subseteq \bigcup \mathcal{X}$; hence if $\alpha \leq_A \beta$ then $\beta \in F \subseteq \bigcup \mathcal{X}$ as F is a filter.

Let $\alpha, \beta \in \bigcup \mathcal{X}$; then there are $F, F' \in \mathcal{X}$ such that $\alpha \in F$ and $\beta \in F'$. By hypothesis there exists an $F'' \in \mathcal{X}$ such that $F \subseteq F'' \supseteq F'$, hence $\alpha, \beta \in F''$ and therefore $\alpha \wedge \beta \in F'' \subseteq \bigcup \mathcal{X}$. □

In view of Lemma 4.3, we write $\bigsqcup \mathcal{Y} = \uparrow(\bigcup \mathcal{Y})$ for the sup of an arbitrary family of filters $\mathcal{Y}$, and $\bigsqcup^\uparrow \mathcal{X} = \bigcup \mathcal{X}$ for a directed family $\mathcal{X}$.

Theorem 4.4 *(Representation theorem). The partial order $(\mathcal{F}_A, \subseteq)$ 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 $(\mathcal{F}_A, \subseteq)$ of filters of some intersection type theory Th_A.*

Proof. To prove the first part note that, by Lemma 4.3.1, we know that $(\mathcal{F}_A, \subseteq)$ is a complete lattice. Let $\mathcal{X} \subseteq \mathcal{F}_A$ be directed; then $\uparrow\alpha \subseteq \bigcup \mathcal{X} = \bigsqcup^\uparrow \mathcal{X}$ and, since $\alpha \in \uparrow\alpha$, there exists $F \in \mathcal{X}$ such that $\alpha \in F$; it follows that $\uparrow\alpha \subseteq F$, since F is upward closed w.r.t. $\leq_A$. This proves that $\uparrow\alpha \in K(\mathcal{F}_A)$; vice versa let $F \in K(\mathcal{F}_A)$ and let $\mathcal{X} = \{\uparrow\alpha \mid \alpha \in F\}$, then $\mathcal{X}$ is directed since if $\alpha, \beta \in F$ then $\alpha \wedge \beta \in F$ and $\uparrow\alpha \subseteq \uparrow\alpha \wedge \beta \supseteq \uparrow\beta$; moreover $F = \bigcup \mathcal{X} = \bigsqcup^\uparrow \mathcal{X}$. By assumption F is compact, hence for some $\alpha \in F$ we have $F \subseteq \uparrow\alpha$, but also $\uparrow\alpha \subseteq F$ as F is a filter, hence $F = \uparrow\alpha$.

By the above we conclude that $K(\mathcal{F}_A) = \{\uparrow\alpha \mid \alpha \in \mathcal{L}_A\}$ which is countable as $\mathcal{L}_A$ is such. Finally from the equality $F = \bigsqcup^\uparrow\{\uparrow\alpha \mid \alpha \in F\} = \bigcup\{\uparrow\alpha \mid \alpha \in F\}$ we conclude that $(\mathcal{F}_A, \subseteq)$ is ω-algebraic.

To the second part, let $\mathcal{L}_A = \{\alpha_d \mid d \in K(A)\}$ (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 $\alpha_d \wedge_A \alpha_e =_A \alpha_{d\sqcup e}$, where $d \sqcup e \in K(A)$ if $d, e \in K(A)$, and that $\omega_A =_A \alpha_\perp$, we have that Th_A is an intersection type theory and that $\mathcal{L}_A/_{\leq_A}$ is isomorphic to $K^{op}(A)$ ordered by the opposite to the ordering of A. From this, it follows that $\mathcal{F}_A$ is isomorphic to the ideal completion of $K(A)$, which in turn is isomorphic to A by algebraicity. □

Remark 4.5. Theorem 4.4 suggests a different interpretation of types as compact points in $K^{\mathrm{op}}(A) \cong K(\mathcal{F}_A)$, that is α is interpreted as the principal filter $\uparrow\alpha$, where we observe that $\uparrow\alpha \subseteq \uparrow\beta$ holds if and only if $\beta \leq_A \alpha$, namely $\alpha \leq_A^{\mathrm{op}} \beta$. To reconcile the two interpretations, let us set $[\![\alpha]\!]^{\mathcal{F}} = \{F \in \mathcal{F}_A \mid \alpha \in F\}$; then

$$[\![\alpha]\!]^{\mathcal{F}} \subseteq [\![\beta]\!]^{\mathcal{F}} \iff \alpha \in \uparrow\beta \iff \uparrow\beta \subseteq \uparrow\alpha \iff \alpha \leq_A \beta$$

On the other hand

$$[\![\alpha \wedge \beta]\!]^{\mathcal{F}} = \{F \in \mathcal{F}_A \mid \alpha \wedge \beta \in F\} = \{F \in \mathcal{F}_A \mid \alpha, \beta \in F\} = [\![\alpha]\!]^{\mathcal{F}} \cap [\![\beta]\!]^{\mathcal{F}}$$

and $[\![\omega_A]\!]^{\mathcal{F}} = \{F \in \mathcal{F}_A \mid \omega_A \in F\} = \mathcal{F}_A$.

The above theorem substantiates the claim that intersection type theories are the *logic* of the domains in ω-**Alg**. In particular, the second part of the proof 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. Therefore, to obtain a finer description of the model in Theorem 3.6, we seek theories Th_D and Th_S such that:

$$\mathcal{F}_D \cong \mathcal{F}_D \to \mathcal{F}_S \to (\mathcal{F}_D \times \mathcal{F}_S)_\perp \tag{6}$$

The first step is to show how the functors involved in the equation above, namely the lifting, the product, and the (continuous) function space can be put in correspondence with the construction of new theories out of the theories which determine the domains combined by the functors. Building over [2], where the larger category of 2/3-SFP domains is considered, we specialize the definition of the respective theories to the case of ω-**Alg**.

Definition 4.6. Suppose that the theories Th_A and Th_B are given, then for $\alpha \in \mathcal{L}_A$ and $\beta \in \mathcal{L}_B$ define:

$$\begin{array}{lll}
\mathcal{L}_{A_\perp} & \alpha_\perp & ::= \alpha \mid \alpha_\perp \wedge \alpha'_\perp \mid \omega_{A_\perp} \\
\mathcal{L}_{A\times B} & \pi & ::= \alpha \times \beta \mid \pi \wedge \pi' \mid \omega_{A\times B} \\
\mathcal{L}_{A\to B} & \phi & ::= \alpha \to \beta \mid \phi \wedge \phi' \mid \omega_{A\to B} \\
\mathcal{L}_{A^{\mathbf{L}}} & \sigma & ::= \langle \ell : \alpha \rangle \mid \sigma \wedge \sigma' \mid \omega_{A^{\mathbf{L}}}
\end{array}$$

Then, we define the following sets of axioms:

1. $\alpha \leq_A \alpha' \Rightarrow \alpha \leq_{A_\perp} \alpha'$.
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. $\omega_{A^{\mathbf{L}}} \leq_{A^{\mathbf{L}}} \langle \ell : \omega_A \rangle$ and all instances of
$$\langle \ell : \alpha \rangle \wedge \langle \ell : \alpha' \rangle \leq_{A^{\mathbf{L}}} \langle \ell : \alpha \wedge \alpha' \rangle$$

Finally, the theories $Th_{A\times B}$, $Th_{A\to B}$ and $Th_{A^{\mathbf{L}}}$ are closed under the rules:

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

In the semantics of λ_{imp} functional application and abstraction play a central role; below we define such operations in the case of domains of filters, toward establishing some properties of them.

Definition 4.7. If $X \in \mathcal{F}_{A\to B}$ and $Y \in \mathcal{F}_A$, we define:

$$X \cdot Y = \{\psi \in \mathcal{L}_B \mid \exists \varphi \in Y \,.\, \varphi \to \psi \in X\}$$

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

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

Lemma 4.8. *If $X \in \mathcal{F}_{A\to B}$ and $Y \in \mathcal{F}_A$ then $X \cdot Y \in \mathcal{F}_B$. 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 $X \cdot Y = \bigsqcup \mathcal{Z}$ where $\mathcal{Z} = \{\uparrow \varphi \cdot \uparrow \psi \mid \varphi \in X \ \& \ \psi \in Y\}$. 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 $\chi \in \uparrow (\psi \to \chi) \cdot \uparrow \psi$. Vice versa if $\chi \in \bigsqcup \mathcal{Z}$ then for finitely many ψ_i, χ_i we have that $\bigwedge_i \chi_i \leq \chi$, $\psi_i \in Y$ and $\psi_i \to \chi_i \in X$. It follows that $\chi_i \in X \cdot Y$ for all i, hence the thesis since $X \cdot Y$ is a filter by the above. □

Lemma 4.9. *If $f : \mathcal{F}_A \to \mathcal{F}_B$ is continuous then $\Lambda(f) \in \mathcal{F}_{A\to B}$. $\Lambda(\cdot)$ is itself continuous and such that:*

$$\Lambda(f) \cdot X = f(X) \qquad \Lambda(\hat{\lambda} Y.(X \cdot Y)) = X$$

Proof. Easy by unfolding definitions and by Lemma 4.8. □

Now we can establish:

Proposition 4.10. *The following are isomorphisms in ω-**Alg**:*

$$\mathcal{F}_{A_\perp} \cong (\mathcal{F}_A)_\perp, \qquad \mathcal{F}_{A\times B} \cong \mathcal{F}_A \times \mathcal{F}_B,$$
$$\mathcal{F}_{A\to B} \cong \mathcal{F}_A \to \mathcal{F}_B, \qquad \mathcal{F}_{A^{\mathbf{L}}} \cong (\mathcal{F}_A)^{\mathbf{L}}.$$

Proof. That $\mathcal{F}_{A_\perp} \cong (\mathcal{F}_A)_\perp$ 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 $\mathcal{F}_A$. $\mathcal{F}_{A\times B} \cong \mathcal{F}_A \times \mathcal{F}_B$ is induced by the continuous extension of the map $\uparrow(\alpha \times \beta) \mapsto (\uparrow \alpha, \uparrow \beta)$, that is clearly invertible. That $\mathcal{F}_{A\to B} \cong \mathcal{F}_A \to \mathcal{F}_B$ is immediate by Lemma 4.9.

Finally, to see that $\mathcal{F}_{A^{\mathbf{L}}} \cong (\mathcal{F}_A)^{\mathbf{L}}$ let us define the maps $F \mapsto \varsigma_F$ from $\mathcal{F}_{A^{\mathbf{L}}}$ to $(\mathcal{F}_A)^{\mathbf{L}}$ and $\varsigma \mapsto F_\varsigma$ from $(\mathcal{F}_A)^{\mathbf{L}}$ to $\mathcal{F}_{A^{\mathbf{L}}}$ by

$$\varsigma_F(\ell) = \bigsqcup \{\uparrow \alpha \mid \langle \ell : \alpha \rangle \in F\} = \{\alpha \mid \langle \ell : \alpha \rangle \in F\}$$

and

$$F_\varsigma = \bigsqcup \{\uparrow \langle \ell : \alpha \rangle \mid \alpha \in \varsigma(\ell)\} = \uparrow \{\langle \ell : \alpha \rangle \mid \alpha \in \varsigma(\ell)\}$$

Then it is routine to prove that these maps are morphisms of ω-**Alg** and inverse each other. □

The next step is to apply Proposition 4.10 to describe the compact elements of $D \cong \mathcal{F}_D$ and of $S \cong \mathcal{F}_S$ and the (inverse of) their orderings. Alas, this cannot be done directly because of the recursive nature of the Equation (6), but it can be obtained by mirroring the inverse limit construction, e.g. along the lines of [4,7]. 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 type theories below simply by mutual induction:

Definition 4.11. Recall that $S = (D_\perp)^{\mathbf{L}}$ and let us abbreviate $C = (D \times S)_\perp$ and $\mathbb{S}_D D = S \to C$; then define the following type languages by mutual induction:

$$\mathcal{L}_D : \quad \delta \ ::= \ \delta \to \tau \mid \delta \wedge \delta' \mid \omega_D$$
$$\mathcal{L}_S : \quad \sigma \ ::= \ \langle \ell : \delta_\perp \rangle \mid \sigma \wedge \sigma' \mid \omega_S \qquad \delta_\perp \in \mathcal{L}_{D_\perp}$$
$$\mathcal{L}_C : \quad \kappa \ ::= \ \delta \times \sigma \mid \kappa \wedge \kappa' \mid \omega_C$$
$$\mathcal{L}_{\mathbb{S}D} : \ \tau \ ::= \ \sigma \to \kappa \mid \tau \wedge \tau' \mid \omega_{\mathbb{S}D}$$

Then the respective itt's $Th_D = (\mathcal{L}_D, \leq_D)$, $Th_S = (\mathcal{L}_S, \leq_S)$, $Th_C = (\mathcal{L}_C, \leq_C)$ and $Th_{\mathbb{S}D} = (\mathcal{L}_{\mathbb{S}D}, \leq_{\mathbb{S}D})$ are defined according to Definition 4.6.

Since $C = (D \times S)_\perp$, the language $\mathcal{L}_{D\times S}$ (not explicitly mentioned in Definition 4.11) is a proper subset of $\mathcal{L}_C$; roughly speaking the difference is arbitrary intersections of ω_C that are all equated to each other in Th_C and strictly greater than any type in $\mathcal{L}_{D\times S}$.

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 $\delta' \to (\sigma' \to (\delta'' \times \sigma''))$.

The following definition will be essential in the technical development:

Definition 4.12. For any $\sigma \in \mathcal{L}_S$ define $dom(\sigma) \subseteq \mathbf{L}$ by:

$$\begin{array}{rcl} dom(\langle \ell : \delta \rangle) & = & \begin{cases} \{\ell\} & \text{if } \delta \neq_{D_\perp} \omega_{D_\perp} \\ \emptyset & \text{otherwise} \end{cases} \\ dom(\sigma \wedge \sigma') & = & dom(\sigma) \cup dom(\sigma') \\ dom(\omega_S) & = & \emptyset \end{array}$$

The set $dom(\sigma)$, which is finite, is computable because it is decidable whether $\delta \neq_{D_\perp} \omega_{D_\perp}$.

Lemma 4.13. *For any $\delta \in \mathcal{L}_{D_\perp}$ it is decidable whether $\delta \neq_{D_\perp} \omega_{D_\perp}$.*

Proof. let $\theta : \mathcal{L}_{D_\perp} \to \mathbf{2}$, where $\mathbf{2} = \{0 \sqsubset 1\}$ is the two points lattice, be defined by:

$$\theta(\delta) = 0 \quad \text{if } \delta \in \mathcal{L}_D, \quad \theta(\omega_{D_\perp}) = 1 \quad \text{and} \quad \theta(\delta \wedge \delta') = \theta(\delta) \sqcap \theta(\delta')$$

where $\sqcap$ is the meet w.r.t. $\sqsubseteq$. Then θ is a total and computable function such that $\delta \neq_{D_\perp} \omega_{D_\perp}$ if and only if $\theta(\delta) = 0$. Indeed, by induction over the definition of $\leq_{D_\perp}$, we have that

$$\delta \leq_{D_\perp} \delta' \Rightarrow \theta(\delta) \sqsubseteq \theta(\delta') \qquad (*)$$

Then

$$\begin{array}{lcll} \delta =_{D_\perp} \omega_{D_\perp} & \Longrightarrow & \delta \wedge \omega_{D_\perp} =_{D_\perp} \omega_{D_\perp} & \text{since } \omega_{D_\perp} \text{ is the top} \\ & \Longrightarrow & \theta(\delta \wedge \omega_{D_\perp}) = \theta(\omega_{D_\perp}) = 1 & \text{by } (*) \\ & \Longrightarrow & \theta(\delta) = 1 & \text{since } \theta(\delta \wedge \omega_{D_\perp}) = \theta(\delta) \sqcap 1 \end{array}$$

By contraposition it follows that if $\theta(\delta) = 0$ then $\delta \neq_{D_\perp} \omega_{D_\perp}$.

Vice versa if $\delta \neq_{D_\perp} \omega_{D_\perp}$ then $\delta <_{D_\perp} \omega_{D_\perp}$; then we show that $\theta(\delta) = 0$ by induction over δ. If $\delta \equiv \omega_D$ or $\delta \equiv \delta' \to \tau$ then $\delta \in \mathcal{L}_D$ hence $\theta(\delta) = 0$. If $\delta \equiv \delta_1 \wedge \delta_2$ then by hypothesis that $\delta_1 \wedge \delta_2 <_{D_\perp} \omega_{D_\perp}$, hence there is $i = 1, 2$ such that $\delta_i <_{D_\perp} \omega_{D_\perp}$; by induction $\theta(\delta_i) = 0$ so that $\theta(\delta) = \theta(\delta_1) \sqcap \theta(\delta_2) = 0$. □

The computability of the finite set $dom(\sigma)$ implies that $\ell \notin dom(\sigma)$ is decidable, which will be the side condition of rule (*set*) in Fig. 1, Section 5.

Remark 4.14. Observe that the only constants used in Definition 4.11 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, $\omega_S \to \omega_C = \omega_{\mathbb{S}D}$.

However, none of the theories above is trivial, because of the strict inequalities:

(i) $\langle \ell : \omega_D \rangle <_S \omega_S$, for any $\ell \in \mathbf{L}$;
(ii) $\omega_D \times \omega_S <_S \omega_C$;
(iii) $\omega_S \to \omega_D \times \omega_S <_{\mathbb{S}D} \omega_S \to \omega_C =_{\mathbb{S}D} \omega_{\mathbb{S}D}$.

Therefore, $\uparrow \omega_S \subset \uparrow \bigwedge_{i \in I} \langle \ell_i : \omega_D \rangle$, for any non empty I, and $\uparrow \omega_C \subset \uparrow (\omega_D \times \omega_S)$, that is $\uparrow \omega_C$ corresponds to the new bottom element added to $\mathcal{F}_{D \times S} \cong \mathcal{F}_D \times \mathcal{F}_S$ in $\mathcal{F}_C = \mathcal{F}_{(D \times S)_\perp} \cong (\mathcal{F}_D \times \mathcal{F}_S)_\perp$.

Theorem 4.15. *The theories Th_D and Th_S induce the filter domains $\mathcal{F}_D$ and $\mathcal{F}_S$ which satisfy Equation (6).*

Proof. By Proposition 4.10 we have

$$\mathcal{F}_{D \to \mathbb{S}D} \cong \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D} \cong \mathcal{F}_D \to \mathcal{F}_S \to (\mathcal{F}_D \times \mathcal{F}_S)_\perp$$

where $\mathcal{F}_S = \mathcal{F}_{(D_\perp)\iota}$. Then the thesis follows since $\mathcal{F}_D = \mathcal{F}_{D \to \mathbb{S}D}$ by construction. □

As it should be clear now, the isomorphism $\mathcal{F}_{D \to \mathbb{S}D} \cong \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$ does not depend on the fact that $\mathcal{F}_D$ is a recursive domain, but only on the fact that Th_D is natural and β-sound in the terminology of [7] (but this was known of the EATS - see [3] Def. 3.3.1 - since [16]).

Remark 4.16. The choice of not having atomic types in $\mathcal{L}_D$ is minimalistic and parallels the analogous definition 5.2.1 of [6], 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 detailed description of $\mathcal{F}_D$ would show, by relating its construction to the solution of Equation (2) in Theorem 3.6, the reason why the ω's suffice is that $\mathcal{F}_D$ is (isomorphic to) the non-trivial initial solution to the domain equation. Adding atomic types ξ to $\mathcal{L}_D$ also leads to a filter model $\mathcal{F}_{D'}$ of λ_{imp}, which is however not isomorphic to $\mathcal{F}_{D'} \to \mathbb{S}(\mathcal{F}_{D'})$. To restore the desired isomorphism it suffices to add axioms $\xi =_{D'} \omega_{D'} \to \omega_S \to (\xi \times \omega_S)$ for all atomic ξ: these correspond to the axioms $\xi = \omega \to \xi$ in [9], which are responsible of obtaining a "natural equated" solution to the equation $D = D \to D$ of Scott's model: see [7].

The λ_{imp} filter model. According to Definition 3.12, to show that $\mathcal{F}_D$ is a λ_{imp}-model it remains to see that $\mathcal{F}_{\mathbb{S}D} \cong \mathbb{S}_D\mathcal{F}_D$ can be endowed with the structure of a monad, which amounts to say that the maps $unit^{\mathcal{F}} : \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$ and $\star^{\mathcal{F}} : \mathcal{F}_{\mathbb{S}D} \times (\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}) \to \mathcal{F}_{\mathbb{S}D}$ are definable in such a way to satisfy the monadic laws. This follows by instantiating Definition 3.3 to filter domains:

$$unit^{\mathcal{F}} X \stackrel{def}{=} \Lambda(\lambda Y \in \mathcal{F}_S.(X, Y)) = \uparrow\{\sigma \to \delta \times \sigma \in \mathcal{L}_{\mathbb{S}D} \mid \delta \in X\} \tag{7}$$

On the other hand, observing that $\mathcal{F}_{\mathbb{S}D} \times (\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}) \to \mathcal{F}_{\mathbb{S}D} \cong \mathcal{F}_{\mathbb{S}D} \times \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$, for all $X \in \mathcal{F}_{\mathbb{S}D}$, $Y \in \mathcal{F}_D$ and $Z \in \mathcal{F}_S$ we expect that:

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

Hence we define $X \star^{\mathcal{F}} Y$ by

$$\uparrow\{\sigma \to \delta'' \times \sigma'' \in \mathcal{L}_{\mathbb{S}D} \mid \exists \delta', \sigma'.\, \sigma \to \delta' \times \sigma' \in X \;\&\; \delta' \to \sigma' \to \delta'' \times \sigma'' \in Y\} \tag{8}$$

Lemma 4.17. *Both $unit^{\mathcal{F}} X$ and $X \star^{\mathcal{F}} Y$ are continuous in X and in X and Y, respectively.*

Proof. By definition, $unit^{\mathcal{F}}(\uparrow\delta) = \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \;\&\; \delta' \in \uparrow\delta\}$; on the other hand if $\delta_0 \leq_D \delta_1$ then $\uparrow\delta_1 \subseteq \uparrow\delta_0$, therefore

$$\begin{aligned} unit^{\mathcal{F}}(\uparrow\delta_1) &= \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \;\&\; \delta' \in \uparrow\delta_1\} \\ &\subseteq \uparrow\{\sigma \to \delta'' \times \sigma \mid \sigma \in \mathcal{L}_S \;\&\; \delta'' \in \uparrow\delta_0\} \\ &= unit^{\mathcal{F}}(\uparrow\delta_0) \end{aligned}$$

Hence $unit^{\mathcal{F}}(\uparrow\delta) \subseteq unit^{\mathcal{F}}(\uparrow(\delta \wedge \delta')) \supseteq unit^{\mathcal{F}}(\uparrow\delta')$ for all δ, δ'; this implies that the family $\{unit^{\mathcal{F}}(\uparrow\delta) \mid \delta \in X\}$ is directed for any filter X. Now

$$\begin{aligned} unit^{\mathcal{F}} X &= unit^{\mathcal{F}}(\textstyle\bigcup_{\delta\in X} \uparrow\delta) && \text{as } X = \bigsqcup^{\uparrow}_{\delta\in X} \uparrow\delta = \bigcup_{\delta\in X} \uparrow\delta \\ &= \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \;\&\; \delta' \in \textstyle\bigcup_{\delta\in X} \uparrow\delta\} && \text{by def. of } unit^{\mathcal{F}} \\ &= \textstyle\bigcup_{\delta\in X} \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \;\&\; \delta' \in \uparrow\delta\} && \text{as } \{\uparrow\delta \mid \delta \in X\} \text{ is directed} \\ &= \textstyle\bigcup_{\delta\in X} unit^{\mathcal{F}}(\uparrow\delta) && \text{by def. of } unit^{\mathcal{F}} \\ &= \textstyle\bigsqcup^{\uparrow}_{\delta\in X} unit^{\mathcal{F}}(\uparrow\delta) && \text{by Lemma 4.3.2 and the above remark} \end{aligned}$$

The proof of the continuity of $X \star^{\mathcal{F}} Y$ is similar. □

The final step is to define the interpretations of $get_\ell : (\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}) \to \mathcal{F}_{\mathbb{S}D} \cong \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$ and $set_\ell : \mathcal{F}_D \times \mathcal{F}_{\mathbb{S}D} \to \mathcal{F}_{\mathbb{S}D}$, which also are derivable from their interpretation into a generic model $\mathcal{D}$.

Definition 4.18. Let $X \in \mathcal{F}_S$, $\ell \in \mathbf{L}$:

$$X \cdot \{\ell\} = \uparrow\{\delta \in \mathcal{L}_D \mid \langle \ell : \delta \rangle \in X\}$$

The above operation represents the application of the "store" X to the location ℓ. Observe that if $X = \uparrow\omega_S$ then there is no $\langle \ell : \delta \rangle \in X$, hence the closure $\uparrow\cdot$ is necessary.

According to Definition 3.12, for any $X \in \mathcal{F}_D$ and $Y \in \mathcal{F}_S$ we must have:

$$get^{\mathcal{F}}_{\ell}(X) \cdot Y = X \cdot (Y \cdot \{\ell\}) \cdot Y$$

where we assume that $\_ \cdot \_$ associates to the left. Now, let $Z = \{\tau \in \mathcal{L}_{\mathbb{S}D} \mid \exists \delta \in \langle \ell : \delta \rangle \in Y \ \& \ \delta \to \tau \in X\}$ 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 $\delta \to \tau = \bigwedge_{i \in I} \delta \to \sigma_i \to \kappa_i \in X$ if and only if $\delta \to \sigma_i \to \kappa_i \in X$ for all $i \in I$. From this we conclude that

$$\begin{array}{rcl} Z \cdot Y & = & \{\kappa \in \mathcal{L}_C \mid \exists \sigma \in Y \mid \sigma \to \kappa \in Z\} \\ & = & \{\kappa \in \mathcal{L}_C \mid \exists \langle \ell : \delta \rangle \wedge \sigma \in Y . \delta \to \sigma \to \kappa \in X\} \end{array}$$

and therefore the appropriate definition of $get^{\mathcal{F}}_{\ell}(X)$ is

$$get^{\mathcal{F}}_{\ell}(X) \stackrel{def}{=} \uparrow \{(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in \mathcal{L}_{\mathbb{S}D} \mid \delta \to (\sigma \to \kappa) \in X\} \tag{9}$$

Similarly, again by Definition 3.12, for any $X \in \mathcal{F}_D$, $Y \in \mathcal{F}_{\mathbb{S}D}$ and $Z \in \mathcal{F}_S$ we expect:

$$set^{\mathcal{F}}_{\ell}(X, Y) \cdot Z = Y \cdot (Z[\ell \mapsto X])$$

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

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

Then

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

and therefore we define:

$$set^{\mathcal{F}}_{\ell}(X, Y) \stackrel{def}{=} \uparrow \{\sigma' \to \kappa \in \mathcal{L}_{\mathbb{S}D} \mid \exists \delta \in X . \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in Y \ \ \& \ \ \ell \notin dom(\sigma')\} \tag{10}$$

Lemma 4.19. *Both $get^{\mathcal{F}}_{\ell}(X)$ and $set^{\mathcal{F}}_{\ell}(X, Y)$ are continuous in X and in X and Y, respectively.*

Proof. We know that $get^{\mathcal{F}}_{\ell}(X) \cdot Y = X \cdot (Y \cdot \{\ell\}) \cdot Y$, hence by Lemma 4.8 to prove that it is continuous in X it suffices to show that $Y \cdot \{\ell\}$ is such in Y, as the composition of continuous functions is continuous. Now

$$\begin{array}{rcl} Y \cdot \{\ell\} & = & \uparrow \{\delta \mid \langle \ell : \delta \rangle \in Y\} \\ & = & \uparrow \{\delta \mid \langle \ell : \delta \rangle \in \bigcup_{\sigma \in Y} \uparrow \sigma\} \\ & = & \bigcup_{\sigma \in Y} \uparrow \{\delta \mid \langle \ell : \delta \rangle \in \uparrow \sigma\} \\ & = & \bigcup_{\sigma \in Y} (\uparrow \sigma \cdot \{\ell\}) \\ & = & \bigsqcup^{\uparrow}_{\sigma \in Y} (\uparrow \sigma \cdot \{\ell\}) \end{array}$$

by directness of $\{\uparrow \sigma \cdot \{\ell\} \mid \sigma \in Y\}$ and Lemma 4.3.

The proof of continuity of $set^{\mathcal{F}}_{\ell}(X, Y)$ is similar, using $set^{\mathcal{F}}_{\ell}(X, Y) \cdot Z = Y \cdot (Z[\ell \mapsto X])$, the continuity of application and the fact that $Z[\ell \mapsto X] = \bigsqcup_{\delta \in X} Z[\ell \mapsto \uparrow \delta]$ that is easily seen. □

Eventually we have:

Theorem 4.20. *The structure $\mathcal{F} = (\mathcal{F}_D, \mathbb{S}, [\![\cdot]\!]^{\mathcal{F}_D}, [\![\cdot]\!]^{\mathcal{F}_{\mathbb{S}D}})$ is a λ_{imp}-model where, for $e \in Env_{\mathcal{F}} = Var \to \mathcal{F}_D$ the interpretations $[\![V]\!]^{\mathcal{F}_D} e \in \mathcal{F}_D$ and $[\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \in \mathcal{F}_{\mathbb{S}D}$ are such that:*

$$\begin{array}{rcl} [\![x]\!]^{\mathcal{F}_D} e & = & e(x) \\ [\![\lambda x.M]\!]^{\mathcal{F}_D} e & = & \uparrow \{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e[x \mapsto \uparrow \delta]\} \\ [\![[V]]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow \{\sigma \to \delta \times \sigma \in \mathcal{L}_{\mathbb{S}D} \mid \delta \in [\![V]\!]^{\mathcal{F}_D} e\} \\ [\![M \star V]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow \{\sigma \to \delta'' \times \sigma'' \mid \exists \delta', \sigma' . \sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \\ & & \qquad \& \ \delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}_D} e\} \\ [\![get_{\ell}(\lambda x.M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow \{(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in \mathcal{L}_{\mathbb{S}D} \mid \sigma \to \kappa \in [\![M]\!]^{\mathcal{F}_D} e[x \mapsto \uparrow \delta]\} \\ [\![set_{\ell}(V, M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow \{\sigma' \to \kappa \mid \exists \delta \in [\![V]\!]^{\mathcal{F}_D} e . \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \\ & & \qquad \& \ \ell \notin dom(\sigma')\} \end{array}$$

Proof. By an easy induction over M, we can show that

$$[\![M]\!]^{\mathcal{F}} e[x \mapsto X] = \bigsqcup\nolimits^{\uparrow}_{\delta \in X} [\![M]\!]^{\mathcal{F}} e[x \mapsto \uparrow \delta]$$

It follows that the function $\lambda\!\!\lambda X \in \mathcal{F}_D.\, [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e[x \mapsto X]$ is continuous. Therefore, applying the clauses of Definition 3.12 to the model $\mathcal{F}$, we get:

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

Then the thesis follows by Equations (7)-(10). □

5. Deriving the type assignment system

In [9] the filter model is a λ-model where validity of typing judgments and derivability in the type assignment system coincide, which is the key for proving the completeness of the type assignment system. Here we revert the process: in Section 4 we have obtained a filter model out of the denotational semantics of the calculus; by extending to λ_{imp} the set theoretic interpretation of types for the ordinary λ-calculus, we get a notion of validity of typing judgments which, when particularized to the model $\mathcal{F}$, determines the assignment system as the axiomatization of the truth in that model. Such axiomatization is implicit in Theorem 4.20 where the interpretation of terms is evidently inductive; as we show in this section, this provides the guidance to derive the typing system in parallel to the inductive definition of the predicate $\varphi \in [\![Q]\!]^{\mathcal{F}} e$ in the theorem, where Q is either a value or a computation and φ is a type of the appropriate language.

Let $\mathcal{D} = (D, [\![\cdot]\!]^D, [\![\cdot]\!]^{\mathbb{S}D})$ be a λ_{imp}-model, and $\Phi : D \xrightarrow{\sim} (D \to \mathbb{S}_D D)$ is the isomorphism in Definition 3.12. Recall that:

$$S = (D_\perp)^{\mathbf{L}} \qquad C = (D \times S)_\perp \qquad \mathbb{S}_D D = S \to C$$

and that $dom(\varsigma) = \{\ell \in \mathbf{L} \mid \varsigma(\ell) \neq \perp_{D_\perp}\}$ for $\varsigma \in S$. In the following A ranges over $D, \mathbb{S}D$ (short for $\mathbb{S}_D D$), S, C.

Definition 5.1 *(Type Interpretation).* Define the maps $[\![\cdot]\!]^A : \mathcal{L}_A \to \wp(A)$ by

$$[\![\omega_A]\!]^A = A \qquad [\![\varphi \wedge \psi]\!]^A = [\![\varphi]\!]^A \cap [\![\psi]\!]^A$$

and

$$\begin{aligned}
[\![\delta \to \tau]\!]^D &= \{d \in D \mid \forall d' \in [\![\delta]\!]^D.\, \Phi(d)(d') \in [\![\tau]\!]^{\mathbb{S}D}\}\\
[\![\langle \ell : \omega_{D_\perp} \rangle]\!]^S &= S\\
[\![\langle \ell : \delta \rangle]\!]^S &= \{\varsigma \in S \mid \ell \in dom(\varsigma)\ \&\ \varsigma(\ell) \in [\![\delta]\!]^D\} \qquad \text{if } \delta \in \mathcal{L}_D\\
[\![\delta \times \sigma]\!]^{D \times S} &= [\![\delta]\!]^D \times [\![\sigma]\!]^S\\
[\![\sigma \to \tau]\!]^{\mathbb{S}D} &= \{g \in \mathbb{S}_D D \mid \forall \varsigma \in [\![\sigma]\!]^S.\, g(\varsigma) \in [\![\tau]\!]^C\}
\end{aligned}$$

As a first consequence of this definition, we have a model of the subtyping relations $\leq_A$ as subset inclusion of type interpretations.

Lemma 5.2. *For $\varphi, \psi \in \mathcal{L}_A$ we have*

$$\varphi \leq_A \psi \Rightarrow [\![\varphi]\!]^A \subseteq [\![\psi]\!]^A$$

Proof. By induction over the definition of $\leq_A$. □

Let us introduce the concept of typing context as follows.

Definition 5.3. A *typing context* Γ is a finite set $\{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ where $n \geq 0$, the x_i's are pairwise distinct, and $\delta_i \in \mathcal{L}_D$ for all i. We set $dom(\Gamma) = \{x_1, \ldots, x_n\}$.

We define the concepts of truth in a model and of validity of the typing $Q : \varphi$ w.r.t. a context Γ as follows.

Definition 5.4. Let $\mathcal{D}$ be a λ_{imp}-model, $e \in Env_D$ and Γ a context; then define

1. $e \models^{\mathcal{D}} \Gamma$ if $e(x) \in [\![\delta]\!]^{\mathcal{D}}$ for all $x : \delta \in \Gamma$
2. $\Gamma \models^{\mathcal{D}} Q : \varphi$ if $[\![Q]\!]^{\mathcal{D}} e \in [\![\varphi]\!]^{\mathcal{D}}$ for all $e \models^{\mathcal{D}} \Gamma$
3. $\Gamma \models Q : \varphi$ if $\Gamma \models^{\mathcal{D}} Q : \varphi$ for all $\mathcal{D}$

When $\Gamma \models^{\mathcal{D}} Q : \varphi$ we say that $Q : \varphi$ is *true* in $\mathcal{D}$ w.r.t. the context Γ; when $\Gamma \models Q : \varphi$ we say that $Q : \varphi$ is *valid* w.r.t. Γ.

Next, we look for a type interpretation in the case of the model $\mathcal{F}$ which we take from [9] (see Remark 4.5):

Definition 5.5. For $A = D, S, C$, and $\mathbb{S}D$, and $\varphi \in \mathcal{L}_A$ define:

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

Such interpretation is a generalization of the natural interpretation of intersection types over an extended type structure [16], which turns out to be a type interpretation in the sense of Definition 5.1.

Proposition 5.6. *The family of mappings $[\![\varphi]\!]^{\mathcal{F}_A}$ is a type interpretation. Moreover:*

1. $[\![\delta \to \tau]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_D \mid \forall Y \in [\![\delta]\!]^{\mathcal{F}}.\ X \cdot Y \in [\![\tau]\!]^{\mathcal{F}}\}$
2. $[\![\langle \ell : \omega_{D_\perp} \rangle]\!]^{\mathcal{F}} = \mathcal{F}_S$
3. $[\![\langle \ell : \delta \rangle]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_S \mid X \cdot \{\ell\} \in [\![\delta]\!]^{\mathcal{F}}\}$ *if* $\delta \in \mathcal{L}_D$
4. $[\![\delta \times \sigma]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_{D\times S} \mid \pi_1(X) \in [\![\delta]\!]^{\mathcal{F}} \ \&\ \pi_2(X) \in [\![\sigma]\!]^{\mathcal{F}}\}$
5. $[\![\sigma \to \kappa]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_{\mathbb{S}D} \mid \forall Y \in [\![\sigma]\!]^{\mathcal{F}}.\ X \cdot Y \in [\![\kappa]\!]^{\mathcal{F}}\}$

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

Proof. The first part is immediate from the definition of type interpretation and of filters.

To see (1) observe that $\uparrow\varphi \in [\![\varphi]\!]^{\mathcal{F}}$. Hence, if $X \in [\![\delta \to \tau]\!]^{\mathcal{F}}$ then $\delta \to \tau \in X$, which implies that $X \cdot \uparrow\delta = \uparrow\tau \in [\![\tau]\!]^{\mathcal{F}}$. Vice versa if $X \cdot Y \in [\![\tau]\!]^{\mathcal{F}}$ for all $Y \in [\![\delta]\!]^{\mathcal{F}}$ then in particular $X \cdot \uparrow\delta = \uparrow\tau \in [\![\tau]\!]^{\mathcal{F}}$, 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.

All the other cases are similar and easier. We just remark that in part (4) $\pi_1(X)$ is a filter: indeed it is nonempty and upward closed because X is such. If $\delta_1, \delta_2 \in \pi_1(X)$ then $\delta_1 \times \sigma_1, \delta_2 \times \sigma_2 \in X$ for some σ_1 and σ_2; then $X \ni (\delta_1 \times \sigma_1) \wedge (\delta_2 \times \sigma_2) = (\delta_1 \wedge \delta_2) \times (\sigma_1 \wedge \sigma_2)$, so that $\delta_1 \wedge \delta_2 \in \pi_1(X)$. The same holds of $\pi_2(X)$. □

As said at the beginning of this section, the type assignment system will be obtained as a complete axiomatization of the truth of typing judgments in $\mathcal{F}$; more precisely we expect to obtain a set of axioms and rules such that

$$\Gamma \vdash Q : \varphi \quad \text{is derivable in the system} \iff \Gamma \models^{\mathcal{F}} Q : \varphi \tag{11}$$

In the first place, we manage to eliminate the universal quantification over the $e \in Env_{\mathcal{F}}$ such that $e \models^{\mathcal{F}} \Gamma$ which is involved in the definition of $\Gamma \models^{\mathcal{F}} Q : \varphi$. To this aim, we endow the set $Env_{\mathcal{F}}$ with a partial order as follows.

Definition 5.7. Over $Env_{\mathcal{F}}$ we define the ordering:

$$e \sqsubseteq e' \Leftrightarrow \forall x \in Var.\ e(x) \subseteq e'(x)$$

Proposition 5.8. *The poset $(Env_{\mathcal{F}}, \sqsubseteq)$ is a cpo, with bottom $e_\perp$ such that $e_\perp(x) = \uparrow\omega_D$ for all $x \in Var$, and for all directed $\mathcal{E} \subseteq Env_{\mathcal{F}}$ there exists the sup $\bigsqcup^{\uparrow}\mathcal{E}$ such that*

$$\forall x \in Var.\ (\bigsqcup\nolimits^{\uparrow}\mathcal{E})(x) = \bigcup_{e\in\mathcal{E}} e(x)$$

Proof. To prove the statement it suffices to check that $\bigsqcup^{\uparrow}\mathcal{E}$ is well defined. Now if $\mathcal{E}$ is directed then for any $e, e' \in \mathcal{E}$ there exists $e'' \in \mathcal{E}$ such that $e \sqsubseteq e'' \sqsupseteq e'$, that is $e(x) \subseteq e''(x) \supseteq e'(x)$ for all $x \in Var$. It follows that $\{e(x) \mid e \in \mathcal{E}\}$ is a directed set of filters for any x, whose sup is $\bigcup_{e\in\mathcal{E}} e(x)$. □

Proposition 5.8 can be restated by saying that $(Env_{\mathcal{F}}, \sqsubseteq)$ is the infinite product of $\mathcal{F}_D$ with itself, indexed over *Var*. As such it is an object of ω-**Alg**, in particular it is algebraic with compact points $e \in K(Env_{\mathcal{F}})$ which are such that for all $x \in Var$, $e(x)$ is compact in $\mathcal{F}_D$, namely $e(x) = \uparrow\varphi$ for some type φ. Among the compact environments, a special role is played by those e_f such that $e_f(x) \neq \uparrow\omega_D$ only for finitely many $x \in Var$, that we call *finite*. As it is easily seen, any environment $e \in K(Env)$ is the directed sup of the set $\{e_f \in K(Env_{\mathcal{F}}) \mid e_f \text{ is finite } \& \ e_f \sqsubseteq e\}$, which then holds for arbitrary $e \in Env$ by algebraicity. Now finite environments can be paired with contexts as follows.

Definition 5.9. Given $\Gamma = \{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ define $e_\Gamma \in Env_{\mathcal{F}}$ by

$$e_\Gamma(x_i) = \uparrow_D \delta_i \qquad \text{and} \qquad e_\Gamma(y) = \uparrow\omega_D \text{ if } y \notin dom(\Gamma)$$

Let us abbreviate $\Gamma(x) = \delta$ if $x : \delta \in \Gamma$ and $\Gamma(x) = \omega_D$, otherwise.

Lemma 5.10. *For all $e \in Env_{\mathcal{F}}$ and context Γ, $e \models^{\mathcal{F}} \Gamma$ if and only if $e_\Gamma \sqsubseteq e$.*

Proof.

$$\begin{array}{lll}
e \models^{\mathcal{F}} \Gamma & \iff \forall x \in Var.\ e(x) \in [\![\Gamma(x)]\!]^{\mathcal{F}} & \text{by Definition 5.4} \\
& \iff \forall x \in Var.\ \Gamma(x) \in e(x) & \text{by Definition 5.5} \\
& \iff \forall x \in Var.\ e_\Gamma(x) = \uparrow\Gamma(x) \subseteq e(x) & \text{since } e(x) \text{ is a filter} \\
& \iff e_\Gamma \sqsubseteq e & \text{by Definition 5.7} \quad \square
\end{array}$$

For any contexts Γ and Γ' we define the context

$$\begin{array}{lll}
\Gamma \wedge \Gamma' & \stackrel{def}{=} & \{x : \delta \in \Gamma \mid x \notin dom(\Gamma')\} \\
& \cup & \{x : \delta' \in \Gamma' \mid x \notin dom(\Gamma)\} \\
& \cup & \{x : \delta \wedge \delta' \mid x : \delta \in \Gamma \ \&\ x : \delta' \in \Gamma'\}
\end{array}$$

In particular $(\Gamma \wedge \Gamma')(x) = \Gamma(x) \wedge \Gamma'(x)$.

Lemma 5.11.

1. *For all $e \in Env_{\mathcal{F}}$ the set $\mathcal{E} = \{e_\Gamma \mid e \models^{\mathcal{F}} \Gamma\}$ is directed and $e = \bigsqcup^{\uparrow} \mathcal{E}$*
2. *For all $Q \in Term$ and directed $\mathcal{E}$, the family of filters $\{[\![Q]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ is directed and*

$$[\![Q]\!]^{\mathcal{F}}(\bigsqcup{}^{\uparrow} \mathcal{E}) = \bigsqcup{}^{\uparrow}_{e \in \mathcal{E}} [\![Q]\!]^{\mathcal{F}} e$$

Proof. Part (1): suppose that $e_\Gamma, e_{\Gamma'} \in \mathcal{E}$; then $e \models^{\mathcal{F}} \Gamma$ and $e \models^{\mathcal{F}} \Gamma'$. By Lemma 5.10 we deduce that $e_\Gamma \sqsubseteq e \sqsupseteq e_{\Gamma'}$. On the other hand

$$\begin{array}{lll}
(e_\Gamma \sqcup e_{\Gamma'})(x) & = & \uparrow\Gamma(x) \sqcup \uparrow\Gamma'(x) \\
& = & \uparrow(\Gamma(x) \wedge \Gamma'(x)) \\
& = & \uparrow(\Gamma \wedge \Gamma')(x)
\end{array}$$

hence $e_\Gamma \sqcup e_{\Gamma'} = e_{\Gamma \wedge \Gamma'}$ by the arbitrary choice of $x \in Var$. Again by Lemma 5.10 we conclude that $e \models^{\mathcal{F}} \Gamma \wedge \Gamma'$, namely $e_{\Gamma \wedge \Gamma'} \in \mathcal{E}$, which is therefore directed.
From the above it follows that $\bigsqcup \mathcal{E} \sqsubseteq e$; to see the inverse it suffices to observe that if $\delta \in e(x)$, then $e_{\{x:\delta\}} \in \mathcal{E}$ so that

$$\delta \in e_{\{x:\delta\}}(x) \subseteq \bigcup_{e_\Gamma \in \mathcal{E}} e_\Gamma(x) = (\bigsqcup{}^{\uparrow} \mathcal{E})(x)$$

for any $x \in Var$, and we conclude that $e \sqsubseteq \bigsqcup \mathcal{E}$.

Part (2): we reason by induction over $Q \in Term$.
Case $Q \equiv x$: then $\{[\![Q]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\} = \{e(x) \mid e \in \mathcal{E}\}$, which is directed by the hypothesis that $\mathcal{E}$ is such. Then

$$[\![Q]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) = (\bigsqcup{}^{\uparrow} \mathcal{E})(x) = \bigcup_{e \in \mathcal{E}} e(x) = \bigsqcup{}^{\uparrow}_{e \in \mathcal{E}} ([\![Q]\!]^{\mathcal{F}} e)$$

Case $Q \equiv \lambda x.M$: let $\mathcal{X} = \{[\![\lambda x.M]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$, and recall that $[\![\lambda x.M]\!]^{\mathcal{F}} e = \uparrow\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e[x \mapsto \uparrow\delta]\}$. Let $F_1, F_2 \in \mathcal{X}$, namely $F_i = [\![\lambda x.M]\!]^{\mathcal{F}} e_i$ for some $e_i \in \mathcal{E}$ and $i = 1, 2$. By directness of $\mathcal{E}$, there exists $e_3 \in \mathcal{E}$ such that $e_1, e_2 \sqsubseteq e_3$; then take $F_3 = [\![\lambda x.M]\!]^{\mathcal{F}} e_3 \in \mathcal{X}$.
W.l.o.g. a non trivial element F_i for $i = 1, 2$ has the shape $\delta_i = \bigwedge_{j \in J_i} \delta_j^{(i)} \to \tau_j^{(i)}$ where $\tau_j^{(i)} \in [\![M]\!]^{\mathcal{F}} e_i[x \mapsto \uparrow\delta_j^{(i)}]$ for all $j \in J_i$. Now the set $\{e_1, e_2, e_3\}$ is directed with sup e_3, so that by induction the set of filters $\{[\![M]\!]^{\mathcal{F}} e_i[x \mapsto \uparrow\delta_j^{(i)}] \mid i = 1, 2, 3\}$ is directed with sup $[\![M]\!]^{\mathcal{F}} e_3[x \mapsto \uparrow\delta_j^{(i)}]$, for all $j \in J_i$. This implies that $\delta_j^{(i)} \to \tau_j^{(i)} \in [\![\lambda x.M]\!]^{\mathcal{F}} e_3$ for all $j \in J_i$, and hence $\delta_i \in [\![\lambda x.M]\!]^{\mathcal{F}} e_3$ as the latter is a filter. We conclude that $F_1, F_2 \subseteq F_3$, so $\mathcal{X}$ is directed. Now

$$\begin{aligned}
[\![Q]\!]^{\mathcal{F}}(\bigsqcup^{\uparrow} \mathcal{E}) &= \uparrow\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}}(\bigsqcup^{\uparrow} \mathcal{E})[x \mapsto \uparrow\delta]\} \\
&= \uparrow\{\delta \to \tau \mid \tau \in \textstyle\bigcup_{e \in \mathcal{E}} [\![M]\!]^{\mathcal{F}} e[x \mapsto \uparrow\delta]\} \\
&\quad \text{by induction and } (\bigsqcup^{\uparrow} \mathcal{E})[x \mapsto \uparrow\delta] = \textstyle\bigsqcup^{\uparrow}_{e \in \mathcal{E}} e[x \mapsto \uparrow\delta] \\
&= \textstyle\bigcup_{e \in \mathcal{E}} \uparrow\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e[x \mapsto \uparrow\delta]\} \\
&\quad \text{since } \mathcal{E} \text{ is directed} \\
&= \textstyle\bigcup_{e \in \mathcal{E}} [\![\lambda x.M]\!]^{\mathcal{F}} e \\
&= \textstyle\bigsqcup^{\uparrow}_{e \in \mathcal{E}} [\![Q]\!]^{\mathcal{F}} e
\end{aligned}$$

Case $Q \equiv M \star V$: then $\{[\![M \star V]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ is directed since $\{[\![M]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ and $\{[\![V]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ are such by induction, $[\![M \star V]\!]^{\mathcal{F}} e = ([\![M]\!]^{\mathcal{F}} e) \star^{\mathcal{F}} ([\![V]\!]^{\mathcal{F}} e)$, and the operator $\star^{\mathcal{F}}$ is continuous by Lemma 4.17, and hence monotonic. Now

$$\begin{aligned}
[\![Q]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) &= [\![M]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) \\
&= (\textstyle\bigsqcup_{e \in \mathcal{E}} [\![M]\!]^{\mathcal{F}} e) \star^{\mathcal{F}} (\bigsqcup_{e' \in \mathcal{E}} [\![V]\!]^{\mathcal{F}} e') && \text{by induction} \\
&= \textstyle\bigsqcup_{e,e' \in \mathcal{E}} ([\![M]\!]^{\mathcal{F}} e) \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e') && \text{by continuity of } _ \star^{\mathcal{F}} _ \\
&= \textstyle\bigsqcup_{e'' \in \mathcal{E}} ([\![M]\!]^{\mathcal{F}} e'') \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e'') && \text{by directness of } \mathcal{E} \\
&= \textstyle\bigsqcup_{e'' \in \mathcal{E}} [\![Q]\!]^{\mathcal{F}} e''
\end{aligned}$$

The remaining cases of $[V]$, $get_\ell(\lambda x.M)$ and $set_\ell(V, M)$ are similar, using the continuity of $unit^{\mathcal{F}}(\_)$, $get_\ell^{\mathcal{F}}(\_)$ and $set_\ell^{\mathcal{F}}(\_, \_)$, respectively. □

The content of part (2) of Lemma 5.11 is that for any $Q \in Term$ the mapping $[\![Q]\!]^{\mathcal{F}} : Env_{\mathcal{F}} \to \mathcal{F}_A$, where A is either D or $\mathbb{S}D$ according to the kind of Q, is continuous, hence monotonic. We use this fact in the proof of the next theorem.

Theorem 5.12. *For any $e \in Env_{\mathcal{F}}$, context Γ, $Q \in Term$ and φ in the appropriate language, we have*

$$\Gamma \models^{\mathcal{F}} Q : \varphi \iff \varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$$

Proof. ($\Rightarrow$) Trivially $e_\Gamma \sqsubseteq e_\Gamma$ so that $e_\Gamma \models^{\mathcal{F}} \Gamma$ by Lemma 5.10. By the hypothesis $\Gamma \models^{\mathcal{F}} Q : \varphi$ we have $[\![Q]\!]^{\mathcal{F}} e_\Gamma \in [\![\varphi]\!]^{\mathcal{F}}$, and hence $\varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$ by Definition 5.5.
($\Leftarrow$) Let $e \models^{\mathcal{F}} \Gamma$, then by Lemma 5.10 $e_\Gamma \sqsubseteq e$; by Lemma 5.11.2 $[\![Q]\!]^{\mathcal{F}}$ is monotonic, so that $[\![Q]\!]^{\mathcal{F}} e_\Gamma \subseteq [\![Q]\!]^{\mathcal{F}} e$. By this the hypothesis $\varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$ implies $\varphi \in [\![Q]\!]^{\mathcal{F}} e$ and hence $[\![Q]\!]^{\mathcal{F}} e \in [\![\varphi]\!]^{\mathcal{F}}$ by Definition 5.5 and we conclude $\Gamma \models^{\mathcal{F}} Q : \varphi$ by the arbitrary choice of e. □

In view of Theorem 5.12, we can obtain the formal system to derive the typing judgments $\Gamma \vdash Q : \varphi$, replacing $\Gamma \models^{\mathcal{F}} Q : \varphi$ by $\varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$ in (11), namely such that

$$\Gamma \vdash Q : \varphi \quad \text{is derivable in the system} \iff \varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$$

The latter is inductively defined in Theorem 4.20 by clauses of the form

$$[\![Q]\!]^{\mathcal{F}} e_\Gamma = \uparrow\{\psi \mid \varphi_1 \in [\![Q_1]\!]^{\mathcal{F}} e_{\Gamma_1} \ \& \ \cdots \ \& \ \varphi_n \in [\![Q_n]\!]^{\mathcal{F}} e_{\Gamma_n}\}$$

which we rephrase into the form

$$\varphi_1 \in [\![Q_1]\!]^{\mathcal{F}} e_{\Gamma_1} \ \& \ \cdots \ \& \ \varphi_n \in [\![Q_n]\!]^{\mathcal{F}} e_{\Gamma_n} \Rightarrow \psi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma \tag{12}$$

and adding the clauses in the left column in Table 1 to take into account the closure operator $\uparrow X$, which is the least filter including the set X.

Table 1
Logical rules.

$\omega \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$	$\dfrac{}{\Gamma \vdash Q : \omega}\ (\omega)$
$\varphi, \psi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma \Rightarrow \varphi \wedge \psi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$	$\dfrac{\Gamma \vdash Q : \varphi \quad \Gamma \vdash Q : \psi}{\Gamma \vdash Q : \varphi \wedge \psi}\ (\wedge)$
$\varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma \ \&\ \varphi \leq \psi \Rightarrow \psi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma$	$\dfrac{\Gamma \vdash Q : \varphi \quad \varphi \leq \psi}{\Gamma \vdash Q : \psi}\ (\leq)$

Table 2
Syntax oriented rules.

$\delta \in \uparrow \delta \ = e_{\Gamma, x:\delta}(x)$ $= [\![x]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$	$\dfrac{}{\Gamma, x : \delta \vdash x : \delta}\ (var)$
$\tau \in [\![M]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$ $\Rightarrow \delta \to \tau \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$	$\dfrac{\Gamma, x : \delta \vdash M : \tau}{\Gamma \vdash \lambda x.M : \delta \to \tau}\ (\lambda)$
$\delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma \ \&\ \sigma \in \mathcal{L}_S$ $\Rightarrow \sigma \to \delta \times \sigma \in [\![[V]]\!]^{\mathcal{F}} e_\Gamma$	$\dfrac{\Gamma \vdash V : \delta}{\Gamma \vdash [V] : \sigma \to \delta \times \sigma}\ (unit)$
$\sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}} e_\Gamma \ \&$ $\delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ $\Rightarrow \sigma \to \delta'' \times \sigma'' \in [\![M \star V]\!]^{\mathcal{F}} e_\Gamma$	$\dfrac{\Gamma \vdash M : \sigma \to \delta' \times \sigma' \quad \Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''}{\Gamma \vdash M \star V : \sigma \to \delta'' \times \sigma''}\ (\star)$
$\sigma \to \kappa \in [\![M]\!]^{\mathcal{F}_D} e_{\Gamma, x:\delta}$ $\Rightarrow (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$ $\in [\![get_\ell(\lambda x.M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma$	$\dfrac{\Gamma, x : \delta \vdash M : \sigma \to \kappa}{\Gamma \vdash get_\ell(\lambda x.M) : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa}\ (get)$
$\delta \in [\![V]\!]^{\mathcal{F}_D} e_\Gamma \ \&$ $\langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma \ \&$ $\ell \notin dom(\sigma')$ $\Rightarrow \sigma' \to \kappa \in [\![set_\ell(V, M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma$	$\dfrac{\Gamma \vdash V : \delta \quad \Gamma \vdash M : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \quad \ell \notin dom(\sigma)}{\Gamma \vdash set_\ell(V, M) : \sigma \to \kappa}\ (set)$

Now the implications (12) are associated with the rules

$$\frac{\Gamma_1 \vdash Q_1 : \varphi_1 \quad \cdots \quad \Gamma_n \vdash Q_n : \varphi_n}{\Gamma \vdash Q : \psi}$$

in the right column of Table 1 and Table 2. Notice that in the second and fifth row of Table 2 we have $e_{\Gamma, x:\delta}$ in place of $e[x \mapsto \uparrow \delta]$ as in Theorem 4.20, however, they are the same environment.

6. Type-semantics and completeness

Collecting all the rules in the right column of Table 1 and Table 2, we get the type assignment system in Fig. 1.

Definition 6.1. (**Intersection type assignment system for** λ_{imp}) The type assignment system for λ_{imp} is defined by the rules in Fig. 1, where in case of rules (ω), $(\wedge)$ and $(\leq)$, the types ω, φ and ψ belong to the languages in accordance with the kind of subject Q.

By construction, the type system enjoys the analogous property of the "Type-semantics theorem" for intersection types and the ordinary λ-calculus (see [11], Thm. 16.2.7). We premise the following lemma, formally proving that the construction in the previous section exactly matches the semantics of terms in the model $\mathcal{F}$.

Lemma 6.2. *Let $Q \in Term$ and Γ be any typing context, then*

$$[\![Q]\!]^{\mathcal{F}} e_\Gamma = \{\varphi \mid \Gamma \vdash Q : \varphi\}$$

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

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

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

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

Fig. 1. Intersection type assignment system for λ_{imp}.

Proof. The proof is based on Theorem 4.20 that characterizes $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma}$ in terms of operations over filters: it will be used without explicit mention below.

The inclusion $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma} \subseteq \{\varphi \mid \Gamma \vdash Q : \varphi\}$ is shown by induction over Q.

Case $Q \equiv x$: then $[\![x]\!]^{\mathcal{F}}_{e_\Gamma} = e_\Gamma(x)$. If $x : \delta \in \Gamma$ for some δ then $e_\Gamma(x) = \uparrow\delta$ and $\Gamma \vdash x : \delta'$ for all $\delta' \geq \delta$ by rules (var) and $(\leq)$. Otherwise $x \notin dom(\Gamma)$ implies that $e_\Gamma(x) = \uparrow\omega_D$ so that $\Gamma \vdash x : \delta$ for any $\delta = \omega_D$ by rules (ω) and $(\leq)$.

Case $Q \equiv \lambda x.M$: then by Definition 4.7 we have

$$\begin{aligned} [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma &= \Lambda(\lambda\!\!\lambda X \in \mathcal{F}_D.\, [\![M]\!]^{\mathcal{F}_{\mathbb{S}_D}} e_\Gamma[x \mapsto X]) \\ &= \uparrow\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e_\Gamma[x \mapsto \uparrow\delta]\} \end{aligned}$$

By definition of the $\uparrow\_$ operator, the filter above consists of intersections $\bigwedge_{i\in I} \delta_i \to \tau_i$ such that $\tau_i \in [\![M]\!]^{\mathcal{F}} e_\Gamma[x \mapsto \uparrow\delta_i]$ for all $i \in I$. Observing that $e_\Gamma[x \mapsto \uparrow\delta_i] = e_{\Gamma, x:\delta_i}$ we have by induction that $\Gamma, x : \delta_i \vdash M : \tau_i$ and therefore $\Gamma \vdash \lambda x.M : \delta_i \to \tau_i$ by rule (λ) for all $i \in I$. Hence $\Gamma \vdash \lambda x.M : \bigwedge_{i\in I} \delta_i \to \tau_i$ by repeated use of rule $(\wedge)$.

Case $Q \equiv [V]$: then $[\![[V]]\!]^{\mathcal{F}} e_\Gamma = \uparrow\{\sigma \to \delta \times \sigma \mid \delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma\}$ by Equation (7). By induction we have $\Gamma \vdash V : \delta$ and the thesis follows by rule $(unit)$.

Case $Q \equiv M \star V$: then, by Equation (8), $[\![M \star V]\!]^{\mathcal{F}} e_\Gamma$ is the filter

$$\uparrow\{\sigma \to \delta'' \times \sigma'' \mid \exists \delta', \sigma'.\, \sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}} e_\Gamma \;\&\; \delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}} e_\Gamma\}$$

As in the case of λ-abstraction, the types in such a filter have the shape $\bigwedge_{i\in I} \sigma_i \to \delta_i'' \times \sigma_i''$ and $\sigma_i \to \delta_i' \times \sigma_i' \in [\![M]\!]^{\mathcal{F}} e_\Gamma$ and $\delta_i' \to \sigma_i' \to \delta_i'' \times \sigma_i'' \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ for all $i \in I$. Therefore by rule $(\star)$ we have $\Gamma \vdash M \star V : \sigma_i \to \delta_i'' \times \sigma_i''$ for all $i \in I$ and we conclude by repeated use of rule $(\wedge)$.

Case $Q \equiv get_\ell(\lambda x.M)$: then by Equation (9) we have that $[\![get_\ell(\lambda x.M)]\!]^{\mathcal{F}} e_\Gamma$ is the filter

$$\uparrow\{(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \mid \delta \to (\sigma \to \kappa) \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma\}$$

Reasoning as above we know that $\delta \to (\sigma \to \kappa) \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$ implies that $\sigma \to \kappa \in [\![M]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$; hence by induction $\Gamma, x : \delta \vdash M : \sigma \to \kappa$, from which the thesis follows by rule (get).

Case $Q \equiv set_\ell(V, M)$: then by Equation (10) we know that $[\![set_\ell(V, M)]\!]^{\mathcal{F}} e_\Gamma$ is the filter

$$\uparrow\{\sigma' \to \kappa \mid \exists \delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma.\, \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in [\![M]\!]^{\mathcal{F}} e_\Gamma \;\&\; \ell \notin dom(\sigma')\}$$

By induction $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \langle \ell : \delta \rangle \wedge \sigma' \to \kappa$; now condition $\ell \notin dom(\sigma')$ allows to apply rule (set), and we are done.

The inclusion $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma} \supseteq \{\varphi \mid \Gamma \vdash Q : \varphi\}$ is proved by induction over the derivation of $\Gamma \vdash Q : \varphi$. The cases of rules (ω), $(\wedge)$ and $(\leq)$ are immediate by the induction hypothesis and the fact that $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma}$ is a filter.

Case (var): then $Q \equiv x$, $\varphi \equiv \delta$ and $x : \delta \in \Gamma$; hence

$$[\![x]\!]^{\mathcal{F}} e_\Gamma = e_\Gamma(x) = \uparrow\delta \ni \delta$$

Case (λ): then $Q \equiv \lambda x.M$, $\varphi \equiv \delta \to \tau$ and the premise is $\Gamma, x : \delta \vdash M : \tau$. By induction $\tau \in [\![M]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$; on the other hand, as observed above, $[\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma = \uparrow\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e_\Gamma[x \mapsto \uparrow\delta]\}$, and $e_\Gamma[x \mapsto \uparrow\delta] = e_{\Gamma, x:\delta}$, hence we conclude that $\delta \to \tau \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$.

Case (*unit*): then $Q \equiv [V]$, $\varphi \equiv \sigma \to \delta \times \sigma$ and the premise is $\Gamma \vdash V : \delta$. By induction $\delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ hence $\sigma \to \delta \times \sigma \in unit^{\mathcal{F}}[\![V]\!]^{\mathcal{F}} e_\Gamma = [\![[V]]\!]^{\mathcal{F}} e_\Gamma$ by Equation (7).

Case ($\star$): then $Q \equiv M \star V$, $\varphi \equiv \sigma \to \delta'' \times \sigma''$ which is derived from the premises $\Gamma \vdash M : \sigma \to \delta' \times \sigma'$ and $\Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''$. By induction $\sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}} e_\Gamma$ and $\delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ so that $\sigma \to \delta'' \times \sigma'' \in [\![M]\!]^{\mathcal{F}} e_\Gamma \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e_\Gamma = [\![M \star V]\!]^{\mathcal{F}} e_\Gamma$ by Equation (8).

Case (*get*): then $Q \equiv get_\ell(\lambda x.M)$, $\varphi \equiv (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$ and the premise is $\Gamma, x : \delta \vdash M : \sigma \to \kappa$. By induction $\sigma \to \kappa \in [\![M]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$ so that, by reasoning as in case (λ), we have that $\delta \to \sigma \to \kappa \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$, from which we conclude $(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in get_\ell^{\mathcal{F}}([\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma) = [\![get_\ell(\lambda x.M)]\!]^{\mathcal{F}} e_\Gamma$ by Equation (9).

Case (*set*): then $Q \equiv set_\ell(V, M)$, $\varphi \equiv \sigma \to \kappa$ and the premises are $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$; also we know that $\ell \notin dom(\sigma)$. By induction $\delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ and $(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in [\![M]\!]^{\mathcal{F}} e_\Gamma$. Then we conclude that $\varphi \equiv \sigma \to \kappa \in set_\ell^{\mathcal{F}}([\![V]\!]^{\mathcal{F}_D} e_\Gamma, [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma) = [\![set_\ell(V, M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma$ by Equation (10). □

We are now in place to state the main theorem of this section.

Theorem 6.3 *(Type semantics). For all $V \in Val$ and $M \in Com$:*

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

Proof. Recall that $e \models^{\mathcal{F}} \Gamma$ if and only if $e_\Gamma \sqsubseteq e$:

$$\begin{array}{rcll}
[\![V]\!]^{\mathcal{F}_D} e & = & [\![V]\!]^{\mathcal{F}_D}(\bigsqcup\{e_\Gamma \mid e \models^{\mathcal{F}} \Gamma\}) & \text{by Lemma 5.11.1} \\
& = & \bigsqcup_{e_\Gamma \sqsubseteq e} [\![V]\!]^{\mathcal{F}_D} e_\Gamma & \text{by Lemma 5.11.2} \\
& = & \bigcup_{e_\Gamma \sqsubseteq e} [\![V]\!]^{\mathcal{F}_D} e_\Gamma & \text{by Lemma 5.11.2} \\
& & & \text{as } \{[\![V]\!]^{\mathcal{F}_D} e_\Gamma \mid e_\Gamma \sqsubseteq e\} \text{ is directed} \\
& = & \bigcup_{e_\Gamma \sqsubseteq e} \{\delta \mid \Gamma \vdash V : \delta\} & \text{by Lemma 6.2} \\
& = & \{\delta \mid \exists \Gamma.\, \Gamma \vdash V : \delta \ \&\ e \models^{\mathcal{F}} \Gamma\} &
\end{array}$$

The proof in case of $[\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e$ is similar. □

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 one typing rule for each syntactical construct in the grammar of the calculus, plus the "logical" rules from Table 1 which are standard in intersection type systems with subtyping since [9].

The immediate consequence of the type-semantics property in [9] is the completeness of the type assignment system. This easily extends to λ_{imp} and to the type assignment system introduced so far. We conclude this section by the completeness theorem for our system.

Theorem 6.4 *(Completeness).*

$$\Gamma \vdash Q : \varphi \Leftrightarrow \Gamma \models Q : \varphi$$

Proof. Part $\Rightarrow$ is shown by induction over $\Gamma \vdash Q : \varphi$. For the $\Leftarrow$ part:

$$\begin{array}{rcll}
\Gamma \models Q : \varphi & \Rightarrow & \Gamma \models^{\mathcal{F}} Q : \varphi & \text{by Definition 5.4 and Theorem 4.20} \\
& \Rightarrow & [\![Q]\!]^{\mathcal{F}} e_\Gamma \in [\![\varphi]\!]^{\mathcal{F}} & \text{since } e_\Gamma \models^{\mathcal{F}} \Gamma \\
& \Rightarrow & \varphi \in [\![Q]\!]^{\mathcal{F}} e_\Gamma & \text{by Definition 5.5} \\
& \Rightarrow & \Gamma \vdash Q : \varphi & \text{by Theorem 6.3} \quad \square
\end{array}$$

7. Discussion and related works

The present paper extends and revises [23], providing a full description of the denotational semantics of λ_{imp}, a detailed construction of the filter model and of the process deriving the type system from the semantics, along with the proofs of the type-semantics and of the completeness theorems.

We have also profited from the present extended version to fix an error in [24,23]. In the former presentation of the theory Th_S we had no distinction among ω_D and $\omega_{D_\perp}$ (as the latter was not included in the type syntax) so that the equation $\omega_S = \langle \ell : \omega_D \rangle$ was derivable. Unfortunately, this falsifies the main theorem of [24], namely the characterization

of convergence. The mismatch arises because there we represent stores by certain store terms that are necessarily finite, hence denoting partial stores; also we consider divergent a term of the form $get_\ell(\lambda x.M)$ when evaluated with a store that is undefined w.r.t. ℓ; this is rather subtle because in a context like $set_\ell(V, get_\ell(\lambda x.M))$ the term will converge as soon as $M[V/x]$ does. To catch this in the model we have to distinguish among the bottom value $\bot_D$, and the new bottom $\bot_{D_\bot}$, represented by the type $\omega_{D_\bot}$, so that we can discriminate the case $\varsigma(\ell) = \bot_{D_\bot}$ modeling the fact that ς is undefined at ℓ, from the case $\varsigma(\ell) = \bot_D$ where ς is defined and holds a value. Here we achieve such a distinction by taking $S = (D_\bot)^{\mathbf{L}}$ instead of $D^{\mathbf{L}}$ as we did in [23].

The calculus and its monadic semantics. But for the operators get_ℓ and set_ℓ, the calculus syntax is the same as in [22], 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.

The algebraic operators get_ℓ and set_ℓ come from Plotkin and Power [38,39,37]. Their definition is essentially the same as in [39], where the store monad in Moggi [34] is generated by the update and lookup operations. In [38] it is proved that the behavior of such operators can be axiomatized by equations such that the monad is determined by the operators themselves under the hypothesis that the set of locations is finite. In our case the locations are infinite so that the monad induced by the algebraic theory in [38] is a proper submonad of ours.

We have borrowed the notation for get_ℓ and set_ℓ from the "imperative λ-calculus" in Chapter 3 of [27], where also a definition of the convergence predicate of a configuration to a result is considered. Such a definition is a particular case of the analogous notion in Dal Lago et al. [21,20] for generic algebraic effects. It is stated in semantic terms, while we preferred the syntactical treatment in the algebra of store terms in [24].

Intersection types and computational effects. Intersection types are an extension of Curry simple types, whose intended meaning is that of predicates of untyped terms. In particular, intersection types embody a form of ad hoc-polymorphism, where one may consider the conjunction of semantically unrelated types. As an instance of Leibnitz's law of identity of indiscernibles, terms can be identified with the set of their properties, namely types. As a consequence, updating the value associated to a location may radically change the types of the store itself.

Our store types are not reference types, and indeed we do not consider the locations among the values, nor we have a construct for dynamic allocation. This makes difficult the comparison to Davis and Pfenning's [25] and to the subsequent Dezani and Ronchi's [18]. In Pfenning's and others work, intersection types are added to the Hindley-Milner type system for ML to enhance type expressivity and to strengthen polymorphism. However, the resulting system is unsound, which forces the restriction of intersection types to values and the loss of subtyping properties. The issue is due to reference types in ML, where the type of a location is its "intrinsic" type in the sense of Reynolds [40]. In contrast, our store types are predicates over the stores, namely "extrinsic" types, telling what are the meanings of the values associated to the locations in the store.

Type and effect systems. A further line of research is to use our type system to investigate a semantic understanding of type and effect systems, introduced in [28] and pursued in [43]. In the insightful paper by Wadler [50] a type and effect judgment $\Gamma \vdash e : A, \varepsilon$ is translated into the ordinary typing $\Gamma \vdash e : T^\varepsilon A$, where T is a monad. This has fostered the application of type systems with monadic types to static analysis for code transformation in Benton et al. [15,14], raising the question of the semantics of the types $T^\varepsilon A$, that has been answered by [32]. See also [36] providing a type interpretation based on indexed (strong) co-monads.

In the papers by Benton and others, the semantics of monadic types with effect decorations is given in terms of PERs that are preserved by read and write operations. Such semantics validates equations that do hold under assumptions about the effects of the equated programs; e.g. only pure terms, neither depending on the store nor causing any mutation, can be evaluated in arbitrary order, or repeated occurrences of the same pure code can be replaced by a reference where the value of the code is stored after computing it just once. Such properties are nicely reflected in our types: if $\lambda x.M$ has type $\delta \to \sigma \to \delta' \times \sigma$, and $dom(\sigma)$ includes all the ℓ occurring in M, then we know that the function represented by $\lambda x.M$ is pure; similarly if $M : \sigma \to \delta \times \sigma$ and $N : \sigma \to \delta' \times \sigma$ then for any $P : \sigma \to \kappa$ both $M; N; P$ and $N; M; P$ will have the same types, and hence the same behavior. In general, this suggests how to encode regions with store types in our system.

Filter model construction. Since [9] 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 study, starting with [16] and continuing with [19,4,1,7]. In the meantime, Abramsky's theory of domain logic in [2] 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.

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 [17,12] 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.

Refinement, essential, and non idempotent intersection types. Another view of intersection types is as refined types of ordinary types (see [2]), so that conjunction makes sense only among subtypes of the same type. This seems in contrast with

the main example in the literature which is the type $(\sigma \wedge (\sigma \to \tau)) \to \tau$ of the term $\lambda x.xx$. However, both σ and $\sigma \to \tau$ are subtypes of $\omega = \omega \to \omega$ in [9], which is the counterpart of Scott's domain equation $D = D \to D$. Here we have the equations $\omega_D = \omega_D \to \omega_{\mathbb{S}D}$ and $\omega_{\mathbb{S}D} = \omega_S \to \omega_C$, representing the solution of the domain equation $D = D \to TD$, where T is (a variant of) the state monad in [34]. The study of type interpretation in the category of models of computational λ-calculi deserves further investigation.

As said before, our type system is inspired by [9], where the intersection types are related by the subtyping relation. It is known that subtyping can be avoided e.g. in the "essential" system in [45], which has syntax directed rules. Compared to the system in [9], the essential intersection type system is equally powerful w.r.t. the characterization of strongly normalizable λ-terms and other similar properties, but it is unrelated to the Scott D_∞ model of the λ-calculus, which is instead isomorphic to the filter model in [9]. In general, the correspondence exploited in [2] among type theories and domains gets lost in case of essential types, describing webbed-models like Engeler's.

Another family of intersection type systems have been introduced by De Carvalho in [17]. Moving from Engeler's model, De Carvalho obtains a system where intersection is not idempotent, that is σ is not a subtype of $\sigma \wedge \sigma$. This provides an intensional type system that is sensible to the temporal complexity of the reduction of terms to normal form. It is a natural development of our work to design a non-idempotent system for the side-effects with higher-order stores.

Towards a categorical perspective. Development of the present work would be a method for synthesizing intersection type systems for a computational λ-calculus with algebraic operators for generic effects in the sense of [39]. Such a process should be described in categorical terms; indeed, while it is well known how to present the denotational semantics of the calculus as a functor into a suitable category of meanings, it has been shown in [35] that a system like ours is itself a particular functor. A natural question is whether the latter functor can be uniformly obtained from the categorical semantics of the calculus.

8. Conclusion

In this paper we presented a type assignment system to study the semantics of an imperative computational λ-calculus equipped with global store and algebraic operators. The system defines the semantics of the untyped calculus.

Deriving the type system from semantics as in Section 5 is the main contribution of our paper. In the present work, we exploit denotational semantics of type systems, in general, but reverse the process from semantics to types. Usually, one starts with a calculus and a type system, looking for semantics and studying its properties. 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 logics catching properties of any computational λ-calculus with operators. We expect that the study of such type systems will be of help in understanding the operational semantics of such calculi, a topic that has been addressed in [21,20], but with different mathematical tools. Indeed, by addressing the investigation in such a way, we smoothly obtain soundness and, possibly, completeness.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] F. Alessi, F. Barbanera, M. Dezani-Ciancaglini, Intersection types and lambda models, Theor. Comput. Sci. 355 (2) (2006) 108–126.
[2] S. Abramsky, Domain theory in logical form, Ann. Pure Appl. Log. 51 (1–2) (1991) 1–77.
[3] R. Amadio, P.-L. Curien, Domains and Lambda-Calculi, Cambridge University Press, 1998.
[4] F. Alessi, M. Dezani-Ciancaglini, F. Honsell, Inverse limit models as filter models, in: Delia Kesner, Femke van Raamsdonk, Joe Wells (Eds.), HOR'04, RWTH Aachen, Aachen, 2004, pp. 3–25.
[5] S. Abramsky, A. Jung, Domain theory, in: Samson Abramsky, Dov M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science (Vol. 3): Semantic Structures, Oxford University Press, Inc., 1994, pp. 1–168.
[6] S. Abramsky, C.-H.L. Ong, Full abstraction in the lazy lambda calculus, Inf. Comput. 105 (2) (1993) 159–267.
[7] F. Alessi, P. Severi, Recursive domain equations of filter models, in: V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, Pavol Návrat, M. Bieliková (Eds.), SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Proceedings, Nový Smokovec, Slovakia, January 19-25, 2008, in: Lecture Notes in Computer Science, vol. 4910, Springer, 2008, pp. 124–135.
[8] H.P. Barendregt, The Lambda Calculus: Its Syntax and Semantics, revised edition, Studies in Logic and the Foundations of Mathematics, vol. 103, North-Holland, 1985.
[9] H.P. Barendregt, M. Coppo, M. Dezani-Ciancaglini, A filter lambda model and the completeness of type assignment, J. Symb. Log. 48 (4) (1983) 931–940.
[10] V. Bono, M. Dezani-Ciancaglini, A tale of intersection types, in: Holger Hermanns, Lijun Zhang, Naoki Kobayashi, Dale Miller (Eds.), LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8–11, 2020, ACM, 2020, pp. 7–20.

[11] H.P. Barendregt, W. Dekkers, R. Statman, Lambda Calculus with Types, Perspectives in Logic, Cambridge University Press, 2013.
[12] A. Bucciarelli, T. Ehrhard, G. Manzonetto, Not enough points is enough, in: J. Duparc, T.A. Henzinger (Eds.), Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, in: Lecture Notes in Computer Science, vol. 4646, Springer, 2007, pp. 298–312.
[13] N. Benton, J. Hughes, E. Moggi, Monads and effects, in: Applied Semantics, International Summer School, APPSEM 2000, in: Lecture Notes in Computer Science, vol. 2395, Springer, 2002, pp. 42–122.
[14] N. Benton, A. Kennedy, L. Beringer, M. Hofmann, Relational semantics for effect-based program transformations: higher-order store, in: A. Porto, F.J. López-Fraguas (Eds.), Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, Coimbra, Portugal, September 7–9, 2009, ACM, 2009, pp. 301–312.
[15] N. Benton, A. Kennedy, M. Hofmann, L. Beringer, Reading, writing and relations, in: Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Proceedings, Sydney, Australia, November 8–10, 2006, in: Lecture Notes in Computer Science, vol. 4279, Springer, 2006, pp. 114–130.
[16] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, G. Longo, Extended type structures and filter lambda models, in: G. Lolli, G. Longo, A. Marcja (Eds.), Logic Colloquium 82, North-Holland, Amsterdam, the Netherlands, 1984, pp. 241–262.
[17] D. de Carvalho, Execution time of λ-terms via denotational semantics and intersection types, Math. Struct. Comput. Sci. 28 (7) (2018) 1169–1203.
[18] M. Dezani-Ciancaglini, S. Ronchi Della Rocca, Intersection and reference types, in: Reflections on Type Theory, Lambda Calculus, and the Mind, Radboud University, Nijmegen, 2007, pp. 77–86.
[19] M. Dezani-Ciancaglini, F. Honsell, F. Alessi, A complete characterization of complete intersection-type preorders, ACM Trans. Comput. Log. 4 (1) (2003) 120–147.
[20] U. Dal Lago, F. Gavazzo, Effectful normal form bisimulation, in: Luís Caires (Ed.), Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings, Prague, Czech Republic, April 6-11, 2019, in: Lecture Notes in Computer Science, vol. 11423, Springer, 2019, pp. 263–292.
[21] U. Dal Lago, F. Gavazzo, P.B. Levy, 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, 2017, pp. 1–12.
[22] U. de'Liguoro, R. Treglia, The untyped computational λ-calculus and its intersection type discipline, Theor. Comput. Sci. 846 (2020) 141–159.
[23] U. de'Liguoro, R. Treglia, From semantics to types: the case of the imperative lambda-calculus, in: Ana Sokolova (Ed.), Proceedings of the 37th Conference on Mathematical Foundations of Programming Semantics, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021, in: Electronic Proceedings in Theoretical Computer Science, vol. 351, Open Publishing Association, 2021, pp. 168–183.
[24] U. de'Liguoro, R. Treglia, Intersection types for a λ-calculus with global store, in: N. Veltri, N. Benton, S. Ghilezan (Eds.), PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6–8, 2021, ACM, 2021, pp. 5:1–5:11.
[25] R. Davies, F. Pfenning, Intersection types and computational effects, in: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), ACM, 2000, pp. 198–208.
[26] M. Felleisen, D.P. Friedman, A syntactic theory of sequential state, Theor. Comput. Sci. 69 (3) (1989) 243–287.
[27] F. Gavazzo, Coinductive Equivalences and Metrics for Higher-Order Languages with Algebraic Effects, PhD thesis, University of Bologna, Italy, April 2019.
[28] D.K. Gifford, J.M. Lucassen, Integrating functional and imperative programming, in: Proceedings of the 1986 ACM Conference on LISP and Functional Programming, LFP 1986, August 4-6, 1986, Cambridge, Massachusetts, USA, ACM, 1986, pp. 28–38.
[29] M. Hyland, J. Power, Discrete Lawvere theories and computational effects, Theor. Comput. Sci. 366 (1–2) (2006) 144–162.
[30] M. Hyland, J. Power, The category theoretic understanding of universal algebra: Lawvere theories and monads, Electron. Notes Theor. Comput. Sci. 172 (2007) 437–458.
[31] J. Roger Hindley, J.P. Seldin, Lambda-Calculus and Combinators: An Introduction, 2nd edition, Cambridge University Press, 2008.
[32] S. Katsumata, Parametric effect monads and semantics of effect systems, in: Suresh Jagannathan, Peter Sewell (Eds.), The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014, ACM, 2014, pp. 633–646.
[33] E. Moggi, Computational Lambda-Calculus and Monads, Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland, 1988.
[34] E. Moggi, Notions of computation and monads, Inf. Comput. 93 (1) (1991) 55–92.
[35] P.-A. Melliès, N. Zeilberger, Functors are type refinement systems, in: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, ACM, 2015, pp. 3–16.
[36] D.A. Orchard, T. Petricek, A. Mycroft, The semantic marriage of monads and effects, CoRR, arXiv:1401.5391, 2014.
[37] J. Power, Generic models for computational effects, Theor. Comput. Sci. 364 (2) (2006) 254–269.
[38] G.D. Plotkin, J. Power, Notions of computation determine monads, in: FOSSACS 2002, in: Lecture Notes in Computer Science, vol. 2303, Springer, 2002, pp. 342–356.
[39] G.D. Plotkin, J. Power, Algebraic operations and generic effects, Appl. Categ. Struct. 11 (1) (2003) 69–94.
[40] J.C. Reynolds, An intrinsic semantics of intersection types, in: Electronic Proceedings of 3rd International Workshop Intersection Types and Related Systems (ITRS'04), Turku, Finland, 2000, pp. 269–270.
[41] M.B. Smyth, G.D. Plotkin, The category-theoretic solution of recursive domain equations, SIAM J. Comput. 11 (4) (1982) 761–783.
[42] V. Swarup, U.S. Reddy, E. Ireland, Assignments for applicative languages, in: J. Hughes (Ed.), Functional Programming Languages and Computer Architecture, 5th ACM Conference, Proceedings, Cambridge, MA, USA, August 26-30, 1991, in: Lecture Notes in Computer Science, vol. 523, Springer, 1991, pp. 192–214.
[43] J.-P. Talpin, P. Jouvelot, The type and effect discipline, Inf. Comput. 111 (2) (1994) 245–296.
[44] M. Tofte, Type inference for polymorphic references, Inf. Comput. 89 (1) (1990) 1–34.
[45] S. van Bakel, Intersection type assignment systems, Theor. Comput. Sci. 151 (2) (1995) 385–435.
[46] P. Wadler, The essence of functional programming, in: Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1992, ACM Press, 1992, pp. 1–14.
[47] P. Wadler, Monads for functional programming, in: Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, in: Lecture Notes in Computer Science, vol. 925, Springer, 1995, pp. 24–52.
[48] A.K. Wright, M. Felleisen, A syntactic approach to type soundness, Inf. Comput. 115 (1) (1994) 38–94.
[49] G. Winskel, K.G. Larsen, Using information systems to solve recursive domain equations effectively, in: G. Kahn, D.B. MacQueen, G.D. Plotkin (Eds.), Semantics of Data Types, International Symposium, Proceedings, Sophia-Antipolis, France, June 27-29, 1984, in: Lecture Notes in Computer Science, vol. 173, Springer, 1984, pp. 109–129.
[50] P. Wadler, P. Thiemann, The marriage of effects and monads, ACM Trans. Comput. Log. 4 (1) (2003) 1–32.