Intersection Types for the Computational \(\lambda\)-Calculus

Extended Abstract

Ugo de’Liguoro and Riccardo Treglia

Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy
ugo.deliguoro@unito.it riccardo.treglia@unito.it

The computational \(\lambda\)-calculus was introduced by Moggi [5,6] as a meta-language to describe non functional effects in programming languages via an incremental approach. The basic idea is to distinguish among values of some type \(D\) and computations over such values, the latter having type \(TD\). Semantically \(T\) is a monad, endowing \(D\) with a richer structure such that operations over computations can be seen as algebras of \(T\). Any \(D\) is embedded into \(TD\) and there is a universal way to extend any morphism in \(D \rightarrow TE\) to a morphism in \(TD \rightarrow TE\).

In Wadler’s formulation [7], at the ground of Haskell implementation, a monad is a triple \((T, \text{unit}, \star)\) where \(T\) is a type constructor, and for all types \(D,E\), \(\text{unit}_D : D \rightarrow TD\) and \(\star_{D,E} : TD \times (D \rightarrow TE) \rightarrow TE\) are such that (omitting subscripts and writing \(\star\) as an infix operator):

\[
(unit \, \! d) \star f = fd, \quad a \star \text{unit} = a, \quad (a \star f) \star g = a \star \lambda d. (f \, \! d \star g).
\]

Instances of monads are partiality, exceptions, input/output, store, non determinism, continuations.

Aim of our work is to investigate the monadic approach to effectful functional languages in the untyped case. Much as the untyped \(\lambda\)-calculus can be seen as a calculus with a single type \(D \rightarrow D\), which is interpreted by a reflexive object in a suitable category, the untyped computational \(\lambda\)-calculus \(\lambda_u^c\) has two types: the type of values \(D\) and the type of computations \(TD\). The type \(D\) is a retract of \(D \rightarrow TD\), which is the call-by-value analogous of the reflexive object (see [5], sec. 5). This leads to the following definition:

**Definition 1 (The untyped computational \(\lambda\)-calculus).** The untyped computational \(\lambda\)-calculus, shortly \(\lambda_u^c\), is a calculus of two sorts of expressions:

\[
\begin{align*}
\text{Val} : & \quad V,W ::= x \mid \lambda x.M \quad \text{(values)} \\
\text{Com} : & \quad M,N ::= \text{unit} \, \! V \mid M \star V \quad \text{(computations)}
\end{align*}
\]

where \(x\) ranges over a denumerable set \(\text{Var}\) of variables.
A reduction relation \( \rightarrow \subseteq \text{Com} \times \text{Com} \) is defined as follows:

\[
(\beta_c) \quad \text{unit} \; V \ast (\lambda x. M) \rightarrow M[V/x]
\]

\[
(\ast - \text{red}) \quad M \rightarrow M' \Rightarrow M \ast V \rightarrow M' \ast V
\]

where \( M[V/x] \) denotes the capture avoiding substitution of \( V \) for all free occurrences of \( x \) in \( M \).

Terms of the calculus can be interpreted into any \( D \simeq D \rightarrow TD \) (where we restrict to extensional models for simplicity) via the mappings \([V]_\rho^D \in D\) and \([M]_\rho^TD \in TD\), where \( \rho \in \text{Env}_D = \text{Var} \rightarrow D \) by:

\[
[x]_\rho^D = \rho(x) \quad \text{unit} \; V \ast [V]_\rho^D
\]

\[
[\lambda x. M]_\rho^D = \lambda d \in D. \; [M]_{\rho[x \mapsto d]}^TD \quad [M \ast V]_\rho^TD = [M]_{\rho}^TD \ast [V]_\rho^D
\]

where \( \rho(x \mapsto d)(y) = \rho(y) \) if \( y \neq x \), it is equal to \( d \) otherwise. We therefore dub (extensional) \( T \)-model in a cartesian closed category \( D \) a tuple \((D, T, \Phi, \Psi)\) such that \( T \) is a monad over \( D \) and \( D \simeq D \rightarrow TD \) via the morphisms \( \Phi, \Psi = \Phi^{-1} \).

**Proposition 1.** If \( M \rightarrow N \) then \([M]_\rho^TD = [N]_\rho^TD\) for any \( T \)-model \( D \) and \( \rho \in \text{Env}_D \).

**An intersection type system for \( \lambda^u_c \)**

To study \( T \)-models we use intersection types, because they are at the same time a formal system to reason on terms and a tool to bridge reduction and operational semantics of the calculus to its models. As shown in [3] reasoning over generic monads is challenging, and indeed a major issue of the present work is to complement Dal Lago’s and others contributions by Coppo-Dezani approach to the study of Scott’s \( D_\infty \) models of the untyped \( \lambda \)-calculus.

Let \( \text{TypeVar} \) be a countable set of type variables, ranged over by \( \alpha \); then we define the following languages of types via the grammar:

\[
\begin{align*}
\text{ValType} : & \quad \delta ::= \alpha \mid \delta \rightarrow \tau \mid \delta \land \delta \mid \omega_V \quad \text{(value types)} \\
\text{ComType} : & \quad \tau ::= T\delta \mid \tau \land \tau \mid \omega_C \quad \text{(computation types)}
\end{align*}
\]

Over types we consider the preorders \( \leq_V \) and \( \leq_C \) making \( \land \) into a meet operator and such that:

\[
\begin{align*}
\delta \leq_V \omega_V & \quad (\delta \rightarrow \tau) \land (\delta \rightarrow \tau') \leq_V \delta \rightarrow (\tau \land \tau') \\
\delta \leq_C \omega_C & \quad T\delta \land T\delta' \leq_C T(\delta \land \delta') \\
\omega_V \leq_V \omega_V & \quad \omega_V \rightarrow \omega_C
\end{align*}
\]

Now we are ready to define the intersection type assignment for \( \lambda^u_c \) and the generic monad \( T \):
Definition 2 (Type assignment). A basis is a finite set of typings $\Gamma = \{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ with pairwise distinct variables $x_i$, whose domain is the set $\text{dom}(\Gamma) = \{x_1, \ldots, x_n\}$. A basis determines a function from variables to types such that $\Gamma(x) = \delta$ if $x : \delta \in \Gamma$, $\Gamma(x) = \omega$ otherwise.

A judgment is an expression of either shapes $\Gamma \vdash V : \delta$ or $\Gamma \vdash M : \tau$. It is derivable if it is the conclusion of a derivation according to the rules:

- $\Gamma, x : \delta \vdash \delta \rightarrow \tau$
- $\Gamma \vdash \lambda x. M : \delta \rightarrow \tau$
- $\Gamma \vdash \text{unit} V : T\delta$
- $\Gamma \vdash M \ast V : \tau$

where $\Gamma, x : \delta = \Gamma \cup \{x : \delta\}$ with $x : \delta \notin \Gamma$, and the rules:

- $\Gamma \vdash P : \sigma \rightarrow \tau$
- $\Gamma \vdash P : \sigma \rightarrow \tau$
- $\Gamma \vdash P : \sigma \rightarrow \tau$
- $\Gamma \vdash P : \sigma \rightarrow \tau$

where either $P \in \text{Val}$, $\omega \equiv \omega_V$, $\sigma, \sigma' \in \text{ValType}$ and $\leq = \leq_V$ or $P \in \text{Com}$, $\omega \equiv \omega_C$, $\sigma, \sigma' \in \text{ComType}$ and $\leq = \leq_C$.

Then by a standard technique, that is by proving suitable Generation and Substitution Lemmas, we establish:

Theorem 1 (Subject reduction). $\Gamma \vdash M : \tau \land M \rightarrow N \Rightarrow \Gamma \vdash N : \tau$.

Type assignment and $T$-models

As a first step we interpret types as certain subsets of $D$ and $TD$, according to the sorts $\text{ValType}$ and $\text{ComType}$ respectively. Let $(D, T, \Phi, \Psi)$ be a $T$-model and $d, d' \in D$, we abbreviate $d \cdot d' = \Phi(d)(d')$. Let $\xi \in \text{TypeEnv}_D = \text{TypeVar} \rightarrow 2^D$; then the followings are natural requirements for the type interpretation mappings $\llbracket \cdot \rrbracket^D : \text{ValType} \times \text{TypeEnv}_D \rightarrow 2^D$ and $\llbracket \cdot \rrbracket^{TD} : \text{ComType} \times \text{TypeEnv}_D \rightarrow 2^{TD}$:

- $\llbracket \alpha \rrbracket^D_{\xi} = \xi(\alpha)$
- $\llbracket \delta \rightarrow \tau \rrbracket^{TD}_{\xi} = \{d \in D \mid \forall d' \in [\delta]^{TD}_{\xi} \ d \cdot d' \in [\tau]^{TD}_{\xi}\}$
- $\llbracket \omega_V \rrbracket^D_{\xi} = D$
- $\llbracket \delta \land \delta' \rrbracket^{TD}_{\xi} = [\delta]^{TD}_{\xi} \cap [\delta']^{TD}_{\xi}$
- $\llbracket \omega_C \rrbracket^{TD}_{\xi} = TD$
- $\llbracket \tau \land \tau' \rrbracket^{TD}_{\xi} = [\tau]^{TD}_{\xi} \cap [\tau']^{TD}_{\xi}$

Further we call these interpretations monadic if $\llbracket T\delta \rrbracket^{TD}_{\xi}$ satisfies:

1. $d \in [\delta]^{TD}_{\xi} \Rightarrow \text{unit} d \in [T\delta]^{TD}_{\xi}$
2. $d \in [\delta' \rightarrow T\delta]^{TD}_{\xi}$ & $a \in [T\delta']^{TD}_{\xi} \Rightarrow a \ast d \in [T\delta]^{TD}_{\xi}$

The main problem with monadic interpretations is that the clauses above are not inductive, as they would be if we had types $\omega_V = \omega_V \rightarrow T\omega_V$ and $T\omega_V$ only. However, working in a category of domains and with an $\omega$-continuous monad $T$ we can build a $T$-model $D_\infty = \lim_n D_n$, where $D_0$ is some fixed domain, and $D_{n+1} = |D_n \rightarrow TD_n|$ is such that for all $n$, $D_n \subseteq D_{n+1}$ is an embedding. As a consequence we have $D_\infty \simeq |D_\infty \rightarrow TD_\infty|$. We say that $D_\infty$ is a limit $T$-model.

More importantly with such a $T$-model we can stratify the above clauses by means of approximate type interpretations $[\delta]^{TD}_{D_n} \subseteq D_n$ and $[\tau]^{TD}_{D_n} \subseteq TD_n$, that now can be defined by induction over $n \in \mathbb{N}$. 

Intersection Types for the Computational $\lambda$-Calculus Extended Abstract
Theorem 2. The mappings \( [\delta]_{\xi}^{D} = \lim_{\rightarrow} [\delta]_{\xi}^{D, n} \) and \( [\tau]_{\xi}^{T D, n} = \lim_{\rightarrow} [\tau]_{\xi}^{T D, n} \) are monadic type interpretations. In particular for any \( \xi \in \text{Env}_{D, n} \):

1. \( [\delta \rightarrow \tau]_{\xi}^{D, n} = \{ d \in D_{\infty} \mid \forall d' \in [\delta]_{\xi}^{D, n} \ d(d') \in [\tau]_{\xi}^{T D, n} \} \)

2. \( [\tau^T]_{\xi}^{D, n} = \{ \text{unit} \ d \in D_{\infty} \mid d \in [\delta]_{\xi}^{D, n} \} \cup \{ a \ast d \in D_{\infty} \mid \exists d'. d \in [\delta'] \rightarrow T[\delta']_{\xi}^{D, n} \ & \ a \in [T[\delta']_{\xi}^{T D, n}] \} \)

Now, writing \( \rho, \xi \models^D \Gamma \) if \( \rho(x) \in [\Gamma(x)]_{\xi}^{D} \) for all \( x \in \text{dom}(\Gamma) \), we may set \( \Gamma \models^D V : \delta \ (\Gamma \models^D M : \tau) \) if \( \rho, \xi \models^D \Gamma \) implies \( [V]_{\rho, \xi}^{D} \in [\delta]_{\xi}^{D} \ (\ [M]_{\rho, \xi}^{D} \in [\tau]_{\xi}^{T D}) \). Also for any class \( C \) of \( T \)-models we write \( \Gamma \models^C V : \delta \ (\Gamma \models^C M : \tau) \) if \( \Gamma \models^D V : \delta \ (\Gamma \models^D M : \tau) \) for all \( D \in C \).

Theorem 3 (Soundness). If \( [\delta]_{\xi}^{D} \) and \( [\tau]_{\xi}^{T D} \) are monadic w.r.t. any \( T \)-model \( D \in C \) then

\[ \Gamma \vdash V : \delta \ \Rightarrow \ \Gamma \models^C V : \delta \ \text{ and } \ \Gamma \vdash M : \tau \ \Rightarrow \ \Gamma \models^C M : \tau. \]

In particular, by Theorem 2, we may take \( C \) as the set of limit \( T \)-models.

Completeness and computational adequacy

Toward completeness, we first concentrate on the category \( D \) of \( \omega \)-algebraic lattices, whose objects are known to be presentable as the poset of filters over a meet-semilattice, or equivalently over a preorder whose quotient is such; the \( \omega \) in the name means that the Scott topology of a domain in \( D \) has a countable basis, formed by the upward cones of compact points. Then any axiomatization \( \text{Th} \) of a preorder over a language \( T \) of intersection types making \( \wedge \) into the meet and \( \omega \) the top, will generate such a domain, and vice versa: we call \( D_{\text{Mor}} = F(\text{Th}) \) the domain of filters w.r.t. \( \leq_{\text{Th}} \) ordered by subset inclusion, and \( D_{\text{Th}} \) the theory of the restriction of the order in \( D \) to the compacts \( K(D) \). Therefore \( D_{\text{Th}} = F(\text{Th}) \approx D \) which we abbreviate by \( F_D \) and identify with \( D \) itself.

Let \( \text{Th}_{\text{Val}} = (\text{ValType}, \leq_{\text{Val}}) \) and \( \text{Th}_{\text{Com}} = (\text{ComType}, \leq_{\text{Com}}) \) and set \( D_{\ast} = D_{\text{Th}_{\text{Val}}} \) and \( T D_{\ast} = D_{\text{Th}_{\text{Com}}} \); then \( \text{Th}_{\text{Val}} \) is a continuous EATS (see e.g. [1] ch. 3, where continuity is expressed by condition \( \text{Frefl} \) of Prop. 3.3.18), hence the space of continuous functions \( D_{\ast} \rightarrow T D_{\ast} \) is representable in \( D_{\ast} \), and actually isomorphic to it. On the other hand the theory \( \text{Th}_{\text{Com}} \) is parametric in \( \text{Th}_{\text{Val}} \). More precisely given a type theory \( \text{Th} \) we can use the axioms of \( \text{Th}_{\text{Com}} \) to form a new theory we call \( T(\text{Th}) \); then we can define a mapping \( T \) among objects of \( D \) by \( TD = D_{T(\text{Th})} \) where \( \text{Th} = \text{Th}_{\text{Th}} \).

Theorem 4. Define unit\( D : F_D \rightarrow F_{TD} \) and \( \ast_{F_{TD}, E} : F_{TD} \times F_{TD} \rightarrow F_{TE} \) by:

\[ \text{unit} F_D d = \uparrow\{ T \delta \in T_{TD} \mid \delta \in d \} \quad t \ast_{F_{TD}, E} e = \uparrow\{ \tau \in T_{TE} \mid \exists \delta \rightarrow \tau \in e. T \delta \in t \} \]

Then \( (T, \text{unit} F, \ast F) \) is a monad over \( D \). Hence \( D_{\ast} \) is a \( T \)-model.

Strictly speaking to enforce extensionality of the filter model, \( \text{Th}_{\text{Val}} \) must be extended to the theory \( \text{Th}_{\text{Val}}' \) by adding suitable axioms: see [4] for the precise treatment.
By stratifying types according to the rank map: \( r(\alpha) = r(\omega_V) = r(\omega_C) = 0 \), 
\( r(\sigma \land \sigma') = \max(r(\sigma), r(\sigma')) \), 
\( r(\delta \to \tau) = \max(r(\delta) + 1, r(\tau)) \) and 
\( r(T\delta) = r(\delta) + 1 \), 
and taking \( \leq_n = \leq_{\{\sigma | r(\sigma) \leq n\}} \) (for both \( \leq_V \) and \( \leq_C \)) we obtain theories \( Th_n \) and a chain of domains \( D_n = \mathcal{F}(Th_n) \) such that \( D = \lim_{\leftarrow} D_n \) is a limit \( T \)-model. Consequently, we can extend the proof in [2] to our calculus obtaining:

**Theorem 5 (Completeness).** Let \( C \) be the class of limit \( T \)-models. Then 
\[
\Gamma \models^C V : \delta \Rightarrow \Gamma \vdash V : \delta \quad \text{and} \quad \Gamma \models^C M : \tau \Rightarrow \Gamma \vdash M : \tau.
\]

**Corollary 1 (Subject expansion).** If \( \Gamma \vdash M : \tau \) and \( N \rightarrow M \) then \( \Gamma \vdash N : \tau \).

Finally let \( \text{Term}^0 = \text{Val}^0 \cup \text{Com}^0 \) be the set of closed terms.

**Definition 3.** Let \( \downarrow \subseteq \text{Com}^0 \times \text{Val}^0 \) be the smallest relation satisfying:
\[
\begin{align*}
\text{unit } V & \downarrow V \\
M \downarrow V & \quad N[V/x] \downarrow W \\
M \star \lambda x.N & \downarrow W
\end{align*}
\]

Then it is easily seen that \( M \downarrow V \) if and only if \( M \xrightarrow{*} \text{unit } V \). We abbreviate \( M \downarrow \Leftrightarrow \exists V. M \downarrow V \).

We say that \( \tau \in \text{ComType} \) is non trivial if \( \omega_C \not\leq C \tau \). Then by adapting Tait’s computability technique, we eventually have:

**Theorem 6.** For all \( M \in \text{Com}^0 \) we have:
\[
M \downarrow \Leftrightarrow \exists \tau \text{ non trivial } \vdash M : \tau
\]

**Corollary 2 (Computational Adequacy).** In the model \( D \) we have that 
\[
M \downarrow \Leftrightarrow [M]^{TD} \neq \bot_{TD}.
\]

From the proof of Theorem 6 we learn that the fact that \( T\omega_V \) is not equated to \( \omega_C \) in \( Th_C \) is an essential ingredient; indeed this corresponds to the fact that the generic monad \( T \) is assumed to be non trivial (hence not the identity monad), so that \( TD \not\cong D \). This supports the intuition that a \( T \)-model equating computations to (the image of) values is not computationally adequate w.r.t. weak normal forms.

For details we refer the reader to the full paper [4].
References

1. Amadio, R., Curien, P.L.: Domains and lambda-calculi. Cambridge University Press (1998)
2. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. Journal of Symbolic Logic 48(4), 931–940 (1983)
3. Dal Lago, U., Gavazzo, F., Levy, P.B.: Effectful applicative bisimilarity: Monads, relators, and howe's method. In: Proc. of Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017. pp. 1–12 (2017)
4. de'Liguoro, U., Treglia, R.: Intersection Types for the Computational lambda-Calculus (Jul 2019), https://arxiv.org/abs/1907.05706, unpublished
5. Moggi, E.: Computational Lambda-calculus and Monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland (Oct 1988)
6. Moggi, E.: Notions of Computation and Monads. Information and Computation 93, 55–92 (1991)
7. Wadler, P.: Monads for Functional Programming. In: Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text. Lecture Notes in Computer Science, vol. 925, pp. 24–52. Springer-Verlag (1995)