Filter models: non-idempotent intersection types, orthogonality and polymorphism

Alexis Bernadet[1,2] **and Stéphane Lengrand**[1,3]

1 **École Polytechnique, France**
2 **École Normale Supérieur de Cachan, France**
3 **CNRS, France**

Abstract

This paper revisits models of typed λ-calculus based on filters of intersection types:

By using *non-idempotent* intersections, we simplify a methodology that produces modular proofs of strong normalisation based on filter models. Non-idempotent intersections provide a decreasing measure proving a key termination property, simpler than the reducibility techniques used with idempotent intersections.

Such filter models are shown to be captured by orthogonality techniques: we formalise an abstract notion of *orthogonality model* inspired by classical realisability, and express a filter model as one of its instances, along with two term-models (one of which captures a now common technique for strong normalisation).

Applying the above range of model constructions to Curry-style System F describes at different levels of detail how the *infinite polymorphism* of System F can systematically be reduced to the *finite polymorphism* of intersection types.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Non-idempotent intersections, System F, realisability

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.51

1 Introduction

Intersection types were introduced in [9], extending the simply-typed λ-calculus with a notion of finite polymorphism. This is achieved by a new construct $A \cap B$ in the syntax of types and new typing rules such as the one on the right, where $M : A$ denotes that a term M is of type A.

$$\frac{M : A \quad M : B}{M : A \cap B}$$

One of the motivations was to characterise strongly normalising λ-terms, the property that a λ-term can be typed if and only if it is strongly normalising. Variants of intersection types systems have been studied to characterise other evaluation properties of λ-terms and served as the basis of corresponding semantics [4, 23, 31, 17, 14, 1].

In particular, intersection types were used to build filter models of λ-calculus as early as [4]. For instance, [1] reveals how the notion of intersection type filter can be tuned so that the corresponding filter models identify those λ-terms that are convertible by various restrictions of β- and η-conversion. Here we rather develop the approach of [10] showing how filters of intersection types can be used to produce models of various *source typing systems*; [10] provides a modular proof that λ-terms that are typable in some (dependent) type theory (the source system) are typable in a unique strongly normalising system of intersection types (the target system), and are therefore strongly normalising.


© Alexis Bernadet and Stéphane Lengrand;
licensed under Creative Commons License NC-ND
Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 51–66

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The contributions of this paper are threefold:

A new target system.

First, we show an improvement on the methodology of [10], changing the target system of idempotent intersection types to a target system that uses *non-idempotent* intersection types (which also departs from [1]). In other words we drop the assumption $A \cap A = A$, which corresponds to the understanding of the judgement $M : A \cap B$ as follows: M can be used as data of type A or data of type B. By dropping idempotency the meaning of $M : A \cap B$ is strengthened in that M **will** be used **once** as data of type A and **once** as data of type B.

The benefit of that move is that the strong normalisation of this new target system follows from the fact that typing trees get strictly smaller with every β-reduction. This is significantly simpler than the strong normalisation of the simply-typed λ-calculus and, even more so, of its extension with idempotent intersection types (for which [10] involves reducibility techniques [18, 30]). Strangely enough there is no price to pay for this simplification, as the construction and correctness of the filter models with respect to a source system is not made harder by non-idempotency.

While this improvement concerns any of the source systems treated in [10], we choose to illustrate the methodology with a concrete source system that includes the impredicative features of System F [18], as suggested in the conclusion of [10]. As explained below, this choice is motivated by an original study of polymorphism.

We propose as a target system a variant of the system in [6], which refined with quantitative information the property that a λ-term is strongly normalising if and only if it can be typed: the length of the longest β-reduction sequence starting from a strongly normalising λ-term can be read off its typing tree. That system was inspired by the pioneering work of [20, 27] as well as [12, 13], where these ideas were connected to the tradition of resource and differential λ-calculi [8, 15] and semantics of linear logic [19]. Although from linear logic have emerged typing systems providing control over the complexity of functional programs [2, 3, 22, 16], let us emphasise that no linearity constraint is here imposed and all strongly normalising λ-terms can be typed (including non-linear ones). In this we also depart from the complexity results specific to the simply-typed λ-terms [29, 5].

Orthogonality models.

The second main contribution of this paper is to show how the above methodology can be formalised in the framework of *orthogonality*. Orthogonality underlies linear logic and its models [19] as well as classical realisability [11, 21, 26], and is used to prove properties of proofs or programs [28, 25, 24].

We formalise here a parametric model construction by introducing an abstract notion of *orthogonality model*, which we illustrate with three different instances:

- one instance based on strongly normalising λ-terms, which captures the traditional use of orthogonality to prove strong normalisation (adapted from [28, 24])
- one instance based on λ-terms that are typable with intersection types
- one instance based on filters of intersection types

To our knowledge, this is the first time that some filter models are shown to be captured by orthogonality techniques. Also, the systematic and modular approach offered by the abstract notion of orthogonality model facilitates the comparison of different proof techniques, e.g. while studying polymorphism.

Polymorphism.

The third contribution of this paper is to use the above technology to shed some new light on finite and infinite polymorphism:

System F and its extensions can assign a type $\forall\alpha\mathfrak{A}$ to a λ-term M, a form of *infinite*

polymorphism, as M can be used with any of the infinitely many instances of $\mathfrak{A}$ as its type. Terms that are typed in System F are strongly normalising [18, 30], and strongly normalising terms can be typed with intersections [9], so a direct corollary is that terms that are typed with infinite polymorphism can in fact be typed with finite polymorphism.

Our model constructions analyse this phenomenon at various levels. The finer-grained analysis is obtained from our filter models, as these give some insight on how the typing trees of System F are transformed into typing trees with finite intersections, where the useful instances of System F types have been computed. Similar ideas were investigated in [32].

Section 2 presents a target system $\lambda_\cap$ of non-idempotent intersection types and its basic properties like strong normalisation. In Section 3 we build filters of non-idempotent intersection types, showing how the use of a target system such as $\lambda_\cap$ simplifies the methodology of [10]; full details are given for concrete examples of source systems such as System F. Section 4 presents the abstract notion of orthogonality model, for the source systems already mentioned. Section 5 presents three instances, one of which captures the construction of a filter model by orthogonality techniques. In Section 6 we discuss how the infinite polymorphism of System F is reduced to the finite polymorphism of intersection types, comparing the different models we have built; then we conclude.

2 Non-idempotent intersection types, improved

Our first goal is to show how non-idempotent intersection types simplify the methodology introduced in [10]. We can use for that the system of non-idempotent intersection types of [6], yet we take in this section the opportunity to make this paper self-contained and present a more syntax-directed variant $\lambda_\cap$ that makes the proofs even simpler.

2.1 Grammar of types and properties

▶ **Definition 1** (Types). Intersection types are defined by the following syntax:

$$\begin{array}{lll} F, G, \ldots & ::= & \tau \mid A \to F \qquad F\text{-types} \\ A, B, \ldots & ::= & F \mid A \cap B \qquad A\text{-types} \\ U, V, \ldots & ::= & A \mid \omega \qquad U\text{-types} \end{array}$$

The intersection $U \cap V$ of arbitrary U-types U and V can be defined by extending the intersection of A-types with: $A \cap \omega := A$, $\omega \cap A := A$ and $\omega \cap \omega := \omega$.

Note that we do **not** assume any implicit equivalence between intersection types (such as idempotency, associativity, commutativity).

▶ Remark. For all U and V, we have $U \cap \omega = \omega \cap U$, and if $U \cap V = \omega$ then $U = V = \omega$.

▶ **Definition 2** ($\approx$). We inductively define $U \approx V$ by the following rules:

$$\frac{}{F \approx F} \qquad \frac{}{A \cap B \approx B \cap A} \qquad \frac{A \approx A' \quad B \approx B'}{A \cap B \approx A' \cap B'} \qquad \frac{A \approx B \quad B \approx C}{A \approx C}$$

$$\frac{}{(A \cap B) \cap C \approx A \cap (B \cap C)} \qquad \frac{}{A \cap (B \cap C) \approx (A \cap B) \cap C} \qquad \frac{}{\omega \approx \omega}$$

The intersection types that we use here differ from those of [6], in that the associativity and commutativity (AC) of the intersection $\cap$ are only featured "on the surface" of types,

and not underneath functional arrows $\to$. This will make the typing rules much more syntax-directed, simplifying the proofs of soundness and completeness of typing with respect to the strong normalisation property. More to the point, this approach reduces the use of the AC properties to the only places where they are needed. Interestingly enough, the idea of not using an equational theory beneath functional arrows is suggested in [1], Remark 3, but not investigated. We have the following properties (the proof can be found in [7]):

▶ **Lemma 3.** *For all U, V, W, F, U', V',*

1. *$\approx$ is an equivalence relation.*
2. *If $U \approx \omega$ then $U = \omega$ and if $U \approx F$ then $U = F$.*
3. *$U \cap V \approx V \cap U$ and $(U \cap V) \cap W \approx U \cap (V \cap W)$.*
4. *If $U \approx U'$ and $V \approx V'$ then $U \cap V \approx U' \cap V'$.*
5. *For all U and V, if $U \cap V \approx U$ then $V = \omega$.*

We equip intersection types with a notion of sub-typing:

▶ **Definition 4** ($\subseteq$)**.** We write $U \subseteq V$ if there exists U' such that $U \approx V \cap U'$.

▶ **Lemma 5.** *For all U, U', V, V' :*

1. *$\subseteq$ is a partial pre-order for intersection types, and $U \approx U'$ if and only if $U \subseteq U'$ and $U' \subseteq U$.*
2. *$U \cap V \subseteq U$ and $U \subseteq \omega$*
3. *If $U \subseteq U'$ and $V \subseteq V'$ then $U \cap V \subseteq U' \cap V'$*

2.2 Typing contexts

We now lift those concepts to typing contexts before presenting the typing rules.

▶ **Definition 6** (Contexts)**.** A context Γ is a total map from variables to U-types such that $\mathsf{Dom}(\Gamma) := \{x \mid \Gamma(x) \neq \omega\}$ is finite. The intersection of contexts $\Gamma \cap \Delta$, and the relations $\Gamma \approx \Delta$ and $\Gamma \subseteq \Delta$, are defined point-wise.

By $()$ we denote the context mapping every variable to ω and by $x{:}U$ the context mapping x to U and every other variable to ω.

The special case of $\Gamma \cap \Delta$ when $\mathsf{Dom}(\Gamma)$ and $\mathsf{Dom}(\Delta)$ are disjoint is denoted Γ, Δ.

▶ **Lemma 7** (Properties of contexts)**.** *For all contexts Γ, Γ', Δ, Δ', Γ'',*

1. *$\Gamma \cap () = \Gamma = () \cap \Gamma$ (for instance $\Gamma, x{:}\omega = \Gamma = x{:}\omega, \Gamma$)*
2. *If $\Gamma \cap \Delta = ()$ then $\Gamma = \Delta = ()$ and if $\Gamma \approx ()$ then $\Gamma = ()$*
3. *$\approx$ is an equivalence relation on contexts*
4. *$\Gamma \cap \Delta \approx \Delta \cap \Gamma$ and $(\Gamma \cap \Gamma') \cap \Gamma'' \approx \Gamma \cap (\Gamma' \cap \Gamma'')$*
5. *If $\Gamma \approx \Gamma'$ and $\Delta \approx \Delta'$ then $\Gamma \cap \Gamma' \approx \Delta \cap \Delta'$*
6. *$\Gamma \subseteq \Delta$ if and only if there exists Γ' such that $\Gamma \approx \Delta \cap \Gamma'$.*
7. *$\subseteq$ is a partial pre-order for contexts, and $\Gamma \approx \Delta$ if and only if $\Gamma \subseteq \Delta$ and $\Delta \subseteq \Gamma$.*
8. *$\Gamma \cap \Delta \subseteq \Gamma$*
9. *If $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$ then $\Gamma \cap \Delta \subseteq \Gamma' \cap \Delta'$.*
10. *$(\Gamma, x{:}U) \subseteq \Gamma$, in particular $\Gamma \subseteq ()$.*

2.3 The target system $\lambda_\cap$

We finally present the typing rules for system $\lambda_\cap$, and its basic properties.

▶ **Definition 8** (λ-calculus)**.** Let Λ be the set of λ-terms defined by the grammar

$$M, N ::= \ x \mid \lambda x.M \mid MN$$

$\lambda x.M$ binds x in M, the free variables $fv(M)$ of a term M are defined as usual and terms are considered up to α-equivalence. The reduction rule is β-reduction:

$$(\lambda x.M)\ N \longrightarrow \ \{{}^{N}/_{x}\}M$$

The congruent closure of this rule is denoted $\longrightarrow_\beta$. SN denotes the set of strongly normalising λ-terms (for β-reduction).

$$\frac{}{x:F \vdash x:F} \qquad \frac{\Gamma, x:U \vdash M:F \quad A \subseteq U}{\Gamma \vdash \lambda x.M : A \to F}$$

$$\frac{\Gamma \vdash M:A \to F \quad \Delta \vdash N:A}{\Gamma \cap \Delta \vdash MN:F} \qquad \frac{\Gamma \vdash M:A \quad \Delta \vdash M:B}{\Gamma \cap \Delta \vdash M:A \cap B} \qquad \frac{}{\vdash M:\omega}$$

Figure 1 System $\lambda_\cap$

▶ **Definition 9** (Typability in System $\lambda_\cap$)**.** The judgement $\Gamma \vdash_\cap M:A$ denotes the derivability of $\Gamma \vdash M:A$ with the rules of Fig. 1. We write $\Gamma \vdash^n_\cap M:A$ if there exists a derivation with n uses of the application rule.

Note that the rule deriving $\vdash M:\omega$ does not interfere with the rest of the system as ω is not an A-type. It is only here for convenience to synthetically express some statements and proofs that would otherwise need a verbose case analysis (e.g. Lemma 11).

Only strongly normalising terms can be assigned an A-type by the system (Theorem 13). In fact, all of them can (Theorem 39), see for instance how the example on the side correctly uses the abstraction rule ($A \subseteq \omega$). Owing to non-idempotency, no closed term inhabits the simple type $(\tau{\to}\tau{\to}\tau'){\to}(\tau{\to}\tau')$ (with $\tau \neq \tau'$), but its natural inhabitant $\lambda f.\lambda x.f\ x\ x$ in a simply-typed system can here be given type $(\tau{\to}\tau{\to}\tau'){\to}(\tau \cap \tau{\to}\tau')$.

$$\dfrac{\dfrac{}{x:F, y:\omega \vdash \lambda y.x:F}}{x:F \vdash \lambda y.x:A{\to}F}$$

Finally, note that the introduction rule for the intersection is directed by the syntax of the type: deciding which rule instance is at the root of a derivation tree typing a term with an intersection, is entirely deterministic. We are not aware of any intersection type system featuring this property, which is here a consequence of dropping the implicit AC properties of intersections, and a clear advantage over the system in [6].

▶ **Lemma 10** (Basic properties of $\lambda_\cap$)**.**

1. *If* $\Gamma \vdash^n_\cap M:U \cap V$ *then there exist* Γ_1, Γ_2, n_1, n_2 *such that* $n = n_1 + n_2$, $\Gamma = \Gamma_1 \cap \Gamma_2$, $\Gamma_1 \vdash^{n_1}_\cap M:U$ *and* $\Gamma_2 \vdash^{n_2}_\cap M:V$.
2. *If* $\Gamma \vdash_\cap M:U$, *then* $\mathsf{Dom}(\Gamma) = fv(M)$.
3. *If* $\Gamma \vdash^n_\cap M:U$ *and* $U \approx U'$ *then there exists* Γ' *such that* $\Gamma \approx \Gamma'$ *and* $\Gamma' \vdash^n_\cap M:U'$
4. *If* $\Gamma \vdash^n_\cap M:U$ *and* $U \subseteq V$ *then there exist* m *and* Δ *such that* $m \leq n$, $\Gamma \subseteq \Delta$ *and* $\Delta \vdash^m_\cap M:V$.

2.4 Strong normalisation of $\lambda_\cap$

This is where non-idempotent intersections provide a real advantage over idempotent ones, as every β-reduction strictly reduces the number of application rules in the typing trees. The proofs, easily adapted from those in [6], can be found in [7].

▶ **Lemma 11** (Typing substitutions). *If* $\Gamma, x:U \vdash^n_\cap M:A$ *and* $\Delta \vdash^m_\cap N:U$ *then there exists* Γ' *such that* $\Gamma' \approx \Gamma \cap \Delta$ *and* $\Gamma' \vdash^{n+m}_\cap \{N/x\}M:A$.

▶ **Theorem 12** (Subject Reduction). *If* $\Gamma \vdash^n_\cap M:A$ *and* $M \longrightarrow_\beta M'$ *then there exist* m *and* Δ *such that* $m < n$, $\Gamma \subseteq \Delta$ *and* $\Delta \vdash^m_\cap M':A$.

▶ **Theorem 13** (Strong Normalisation). *If* $\Gamma \vdash_\cap M:A$ *then* $M \in \mathit{SN}$.

The converse is also true (strongly normalising terms can be typed in $\lambda_\cap$), see Theorem 39 and more generally Appendix A (with Subject Expansion, etc.).

3 Building an I-filter model for a source system

In this section we show how to use non-idempotent intersection types to simplify the methodology of [10], which we briefly review here:

The goal is to produce modular proofs of strong normalisation for various *source typing systems*. The problem is reduced to the strong normalisation of a unique *target system* of intersection types, chosen once and for all. This is done by interpreting each λ-term t as the set $[\![t]\!]$ of the intersection types that can be assigned to t in the target system. Two facts then remain to be proved:

1. if t can be typed in the source system, then $[\![t]\!]$ is not empty
2. the target system is strongly normalising

The first point is the only part that is specific to the source typing system: it amounts to turning the interpretation of terms into a filter model of the source typing system. The second point depends on the chosen target system: as [10] uses a system of *idempotent* intersection types (extending the simply-typed λ-calculus), their proof involves the usual reducibility technique [18, 30]. But this is somewhat redundant with point 1 which uses similar techniques to prove the correctness of the filter model with respect to the source system.[1]

In this paper we propose to use *non-idempotent* intersection types for the target system, so that point 2 can be proved with simpler techniques than in [10] while point 1 is not impacted by the move. In practice we propose $\lambda_\cap$ as the target system (that of [6] would work just as well).[2] We now show the details of this alternative.

3.1 I-filters of non-idempotent intersection types

The following filter constructions only involve the syntax of types and are independent from the chosen target system.

▶ **Definition 14** (I-filter).

An I-filter is a set v of A-types such that:

[1] If reducibility techniques are needed for the latter, why not use them on the source system directly (besides formulating a modular methodology)?

[2] Correspondingly, there is no simple way to embed the simply-typed λ-calculus into $\lambda_\cap$, other than considering it as the source system of the present methodology.

  - for all A and B in v we have $A \cap B \in v$
  - for all A and B, if $A \in v$ and $A \subseteq B$ then $B \in v$
- In particular the empty set and the sets of all A-types are I-filters and we write them $\bot$ and $\top$ respectively.
- Let $\mathcal{D}$ be the set of all non-empty I-filters; we call such I-filters *values*.
- Let $\mathcal{E}$ be the set of all I-filters ($\mathcal{E} = \mathcal{D} \cup \{\bot\}$).

While our intersection types differ from those in [10] (in that idempotency is dropped), the stability of a filter under type intersections makes it validate idempotency (it contains A if and only if it contains $A \cap A$, etc). This makes our filters very similar to those in [10], so we can plug-in the rest of the methodology with minimal change.

▶ Remark (Basic properties of I-filters).
1. If $(v_i)_{i\in I}$ is an non empty family of $\mathcal{E}$ then $\bigcap_{i\in I} v_i \in \mathcal{E}$.
2. If v is a set of A-types then there is a smallest $v' \in \mathcal{E}$ such that $v \subseteq v'$ and we write $< v >:= v'$.
3. If v is a set of F-types then $< v >$ is the closure of v under finite intersections.
4. If $v \in \mathcal{E}$ then $v =< \{F \mid F \in v\} >$.
5. If u and v are sets of F-types such that $< u >=< v >$ then $u = v$.

Hence, in order to prove that two I-filters are equal we just have to prove that they contain the same F-types.

I-filters form an applicative structure:

▶ **Definition 15** (Application of I-filters)**.** If u, v are in $\mathcal{E}$ then define

$$u@v := < \{F \mid \exists A \in v, (A \to F) \in u\} >$$

▶ Remark. For all $u \in \mathcal{E}$, $u@\bot = \bot@u = \bot$, and for all $u \in \mathcal{D}$, $\top@u = \top$.

▶ **Definition 16** (Environments and contexts)**.** An *environment* is a map from term variables $x, y, \ldots$ to I-filters. If ρ is an environment and Γ is a context, we say that $\Gamma \in \rho$, or Γ is *compatible with* ρ, if for all x, $\Gamma(x) = \omega$ or $\Gamma(x) \in \rho(x)$.

▶ Remark (Environments are I-filters of contexts). [3] Let ρ be an environment.
1. If $\Gamma \in \rho$ and $\Gamma' \in \rho$ then $\Gamma \cap \Gamma' \in \rho$.
2. If $\Gamma \in \rho$ and Γ' is a context such that $\Gamma \subseteq \Gamma'$ then $\Gamma' \in \rho$.

3.2 Interpretation of terms

The remaining ingredients now involve the target system; we treat here $\lambda_\cap$.

▶ **Definition 17** (Interpretation of terms)**.** If M is a term and ρ is an environment we define

$$[\![M]\!]_\rho := \{A \mid \exists \Gamma \in \rho, \Gamma \vdash_\cap M : A\}$$

▶ Remark. $[\![M]\!]_\rho \in \mathcal{E}$, and therefore $[\![M]\!]_\rho =< \{F \mid \exists \Gamma \in \rho, \Gamma \vdash_\cap M : F\} >$.

▶ **Theorem 18** (Inductive characterisation of the interpretation)**.**
1. $[\![x]\!]_\rho = \rho(x)$
2. $[\![MN]\!]_\rho = [\![M]\!]_\rho @ [\![N]\!]_\rho$

[3] Conversely, if E is an I-filter of contexts then ρ, defined by $\rho(x) = \{\Gamma(x) \neq \omega \mid \Gamma \in E\}$ for all x, is an environment.

3. $[\![\lambda x.M]\!]_\rho @ u = [\![M]\!]_{\rho, x \mapsto u}$ *for any value* u

This theorem (proved in [7]) makes $\lambda_\cap$ a suitable alternative as a target system: the filter models of the source systems treated in [10] can be done with a system of non-idempotent intersection types. While we could develop those constructions, we prefer to cover a new range of source systems: those with second-order quantifiers such as System F, as this will shed a new light on polymorphism.

3.3 Concrete examples for the source system

▶ **Definition 19** (Types and Typing System)**.** Types are built as follows:

$$\mathfrak{A}, \mathfrak{B}, \ldots ::= \ \alpha \mid \mathfrak{A} \to \mathfrak{B} \mid \mathfrak{A} \cap \mathfrak{B} \mid \forall \alpha \mathfrak{A}$$

where α denotes a type variable, $\forall \alpha \mathfrak{A}$ binds α in $\mathfrak{A}$, types are considered modulo α-conversion, and $ftv(\mathfrak{A})$ denotes the free (type) variables of $\mathfrak{A}$.

Typing contexts, denoted $\mathfrak{G}$, $\mathfrak{H}, \ldots$ are partial maps from term variables to types, $(x : \mathfrak{A})$ denotes the map from x to $\mathfrak{A}$, and . $\mathfrak{G}, \mathfrak{H}$ denotes the union of $\mathfrak{G}$ and $\mathfrak{H}$ (as graphs). Fig. 2 shows a collection of well-known typing rules to type pure λ-terms.

$$\frac{}{\mathfrak{G}, x : \mathfrak{A} \vdash x : \mathfrak{A}} \qquad \frac{\mathfrak{G}, x : \mathfrak{A} \vdash M : \mathfrak{B}}{\mathfrak{G} \vdash \lambda x.M : \mathfrak{A} \to \mathfrak{B}} \qquad \frac{\mathfrak{G} \vdash M : \mathfrak{A} \to \mathfrak{B} \quad \mathfrak{G} \vdash N : \mathfrak{A}}{\mathfrak{G} \vdash M\ N : \mathfrak{B}}$$

$$\frac{\mathfrak{G} \vdash M : \mathfrak{A} \quad \mathfrak{G} \vdash M : \mathfrak{B}}{\mathfrak{G} \vdash M : \mathfrak{A} \cap \mathfrak{B}} \qquad \frac{\mathfrak{G} \vdash M : \mathfrak{A} \cap \mathfrak{B}}{\mathfrak{G} \vdash M : \mathfrak{A}} \qquad \frac{\mathfrak{G} \vdash M : \mathfrak{A} \cap \mathfrak{B}}{\mathfrak{G} \vdash M : \mathfrak{B}}$$

$$\frac{\mathfrak{G} \vdash M : \mathfrak{A}}{\mathfrak{G} \vdash M : \forall \alpha \mathfrak{A}}\ \alpha \notin ftv(\mathfrak{G}) \qquad \frac{\mathfrak{G} \vdash M : \forall \alpha \mathfrak{A}}{\mathfrak{G} \vdash M : \{\mathfrak{B}/\alpha\}\mathfrak{A}}$$

Figure 2 Miscellaneous Typing Rules

Let $\mathcal{S}$ be the typing system consisting of an arbitrary subset of the rules. Typability in system $\mathcal{S}$ will be expressed by judgements of the form $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$.

We now build the model $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$, starting with a notion of realisability candidate:

▶ **Definition 20** (Realisability Predicate)**.** A *realisability predicate* is a subset X of $\mathcal{D}$ containing $\top$. We define $\mathsf{TP}(\mathcal{D})$ as the set of realisability predicates.

▶ **Lemma 21** (Shape of realisability predicates)**.**

1. *If* $(X_i)_{i \in I}$ *an non empty family of* $\mathsf{TP}(\mathcal{D})$ *then* $\bigcap_{i \in I} X_i \in \mathsf{TP}(\mathcal{D})$.

2. *If* X *and* Y *in* $\mathsf{TP}(\mathcal{D})$ *then* $X \to Y \in \mathsf{TP}(\mathcal{D})$ *where* $X \to Y$ *is defined as*

$$X \to Y := \ \{u \mid \forall v \in X, u @ v \in Y\}$$

Proof. The only subtle point is the second one: First, for all $v \in X$, $v \neq \bot$ and thus $\top @ v = \top \in Y$. So $\top \in X \to Y$. Second, suppose that $\bot \in X \to Y$. As $X \neq \emptyset$, there is $u \in X$, for which $\bot @ u = \bot \in Y$, which contradicts $Y \in \mathsf{TP}(\mathcal{D})$. ◀

The model $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ consists of the interpretations of terms (Definition 17) and types:

▶ **Definition 22** (Interpretation of types).
Valuations are mappings from type variables to elements of $\mathsf{TP}(\mathcal{D})$.

Given such a valuation σ, the interpretation of types is defined as follows:

$$\begin{array}{lll} [\![\alpha]\!]_\sigma & := & \sigma(\alpha) \\ [\![\mathfrak{A}{\to}\mathfrak{B}]\!]_\sigma & := & [\![\mathfrak{A}]\!]_\sigma {\to} [\![\mathfrak{B}]\!]_\sigma \end{array} \qquad \begin{array}{lll} [\![\mathfrak{A}\cap\mathfrak{B}]\!]_\sigma & := & [\![\mathfrak{A}]\!]_\sigma \cap [\![\mathfrak{B}]\!]_\sigma \\ [\![\forall\alpha\mathfrak{A}]\!]_\sigma & := & \bigcap_{X\in\mathsf{TP}(\mathcal{D})} [\![\mathfrak{A}]\!]_{\sigma,\alpha\mapsto X} \end{array}$$

The interpretation of typing contexts is defined as follows:

$$[\![\mathfrak{G}]\!]_\sigma := \{\rho \mid \forall (x{:}\mathfrak{A}) \in \mathfrak{G}, \rho(x) \in [\![\mathfrak{A}]\!]_\sigma\}$$

Finally we get Adequacy, by a simple induction on derivations, using Theorem 18:

▶ **Lemma 23** (Adequacy Lemma). *If $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, then for all valuations σ and for all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]_\rho \in [\![\mathfrak{A}]\!]_\sigma$.*

▶ **Corollary 24** (Strong normalisation of $\mathcal{S}$). *If $\mathfrak{G} \vdash_{\mathcal{S}} M{:}\mathfrak{A}$ then $M \in \mathsf{SN}$.*

Proof. Applying the previous lemma with σ mapping every type variable to $\{\top\}$ and ρ mapping all term variable to $\top$, we get $[\![M]\!]_\rho \in [\![\mathfrak{A}]\!]_\sigma$, so $[\![M]\!]_\rho \neq \bot$. Hence, M can be typed in $\lambda_\cap$, so $M \in \mathsf{SN}$. ◀

The advantage of non-idempotent intersection types (over idempotent ones) lies in the very last step of the above proof: here the typing trees of $\lambda_\cap$ get smaller with every β-reduction (proof of Theorem 13), while a reducibility technique as in [10] combines yet again an induction on types with an induction on typing trees similar to that in the Adequacy Lemma.

4 Orthogonality models of typed λ-calculi

In this section we show how the above methodology can be integrated to the theory of *orthogonality*, i.e. how this kind of filter model construction can be captured by orthogonality techniques [19, 11, 21, 26]. These techniques are particularly suitable to prove that typed terms satisfy some property [28, 25, 24], the most well-known of which being Strong Normalisation.

For this we define an abstract notion of orthogonality model for a system $\mathcal{S}$ built from the rules of Fig. 2. Our definition thus applies to the simply-typed λ-calculus, the idempotent intersection type system, System F, etc but we could also adapt it with no difficulty to accommodate System F_ω.

Orthogonality techniques and the filter model construction from Section 3 (with the sets $\mathcal{D}$ and $\mathcal{E}$) inspire the notion of orthogonality model below. First we need notations:

▶ Notation. Given a set $\mathcal{D}$, let $\mathcal{D}^*$ be the set of lists of elements of $\mathcal{D}$, with [] representing the empty list and $u :: \overrightarrow{v}$ representing the list of head u and tail $\overrightarrow{v}$.

▶ **Definition 25** (Orthogonality model).
An *orthogonality model* is a 4-tuple $(\mathcal{E}, \mathcal{D}, \perp\!\!\!\perp, [\![\_]\!]\_)$ where

- $\mathcal{E}$ is a set, called the *support*
- $\mathcal{D} \subseteq \mathcal{E}$ is a set of elements called *values*
- $\perp\!\!\!\perp \subseteq \mathcal{D} \times \mathcal{D}^*$ is called the *orthogonality relation*
- $[\![\_]\!]\_$ is a function mapping every λ-term M (typed or untyped) to an element $[\![M]\!]_\rho$ of the support, where ρ is a parameter called *environment* mapping term variables to values.
- the following axioms are satisfied:

(A1) For all ρ, $\overrightarrow{v}$, x, if $\rho(x) \perp\!\!\!\perp \overrightarrow{v}$ then $[\![x]\!]_\rho \perp\!\!\!\perp \overrightarrow{v}$.

(A2) For all ρ, $\overrightarrow{v}$, M_1, M_2, if $[\![M_1]\!]_\rho \perp\!\!\!\perp ([\![M_2]\!]_\rho :: \overrightarrow{v})$ then $[\![M_1\ M_2]\!]_\rho \perp\!\!\!\perp \overrightarrow{v}$.
(A3) For all ρ, $\overrightarrow{v}$, x, M and for all values u, if $[\![M]\!]_{\rho,x\mapsto u} \perp\!\!\!\perp \overrightarrow{v}$ then $[\![\lambda x.M]\!]_\rho \perp\!\!\!\perp (u :: \overrightarrow{v})$.

In fact, $\mathcal{D}$ and $\perp\!\!\!\perp$ are already sufficient to interpret any type A as a set $[\![A]\!]$ of values (see Definition 28 below): if types are seen as logical formulae, we can see this construction as a way of building some of their realisability / set-theoretical models.

There is no notion of computation pertaining to values, but the interplay between the interpretation of terms and the orthogonality relation is imposed by the axioms so that the Adequacy Lemma (which relates typing to semantics) holds:

$$\text{If } \vdash M : A \text{ then } [\![M]\!] \in [\![A]\!]$$

4.1 Interpretation of types and Adequacy Lemma

▶ **Definition 26** (Orthogonal).
- If $X \subseteq \mathcal{D}$ then let $X^\perp := \{\overrightarrow{v} \in \mathcal{D}^* \mid \forall u \in X, u \perp\!\!\!\perp \overrightarrow{v}\}$
- If $Y \subseteq \mathcal{D}^*$ then let $Y^\perp := \{u \in \mathcal{D} \mid \forall \overrightarrow{v} \in Y, u \perp\!\!\!\perp \overrightarrow{v}\}$

▶ Remark. If $X \subseteq \mathcal{D}$ or $X \subseteq \mathcal{D}^*$ then $X \subseteq X^{\perp\perp}$ and $X^{\perp\perp\perp} = X^\perp$.

▶ **Definition 27** (Lists and Cons construct). If $X \subseteq \mathcal{D}$ and $Y \subseteq \mathcal{D}^*$, then define

$$X :: Y := \{u :: \overrightarrow{v} \mid u \in X, \overrightarrow{v} \in Y\}$$

▶ **Definition 28** (Interpretation of types).
Mappings from type variables to subsets of $\mathcal{D}^*$ are called *valuations*.

Given such a valuation σ, the interpretation of types is defined as follows:

$$\begin{array}{llll} [\alpha]_\sigma & := \sigma(\alpha) & [\mathfrak{A} \cap \mathfrak{B}]_\sigma & := [\mathfrak{A}]_\sigma \cup [\mathfrak{B}]_\sigma \\ [\mathfrak{A}\to\mathfrak{B}]_\sigma & := [\![\mathfrak{A}]\!]_\sigma :: [\mathfrak{B}]_\sigma & [\forall\alpha\mathfrak{A}]_\sigma & := \bigcup_{Y\subseteq\mathcal{D}^*} [\mathfrak{A}]_{\sigma,\alpha\mapsto Y} \end{array}$$
$$[\![\mathfrak{A}]\!]_\sigma := [\mathfrak{A}]_\sigma^\perp$$

The interpretation of typing contexts is defined as follows:

$$[\![\mathfrak{G}]\!]_\sigma := \{\rho \mid \forall (x:\mathfrak{A}) \in \mathfrak{G}, \rho(x) \in [\![\mathfrak{A}]\!]_\sigma\}$$

▶ Remark. Note that $[\{\mathfrak{B}/\alpha\}\mathfrak{A}]_\sigma = [\mathfrak{A}]_{\sigma,\alpha\mapsto[\mathfrak{B}]_\sigma}$ and $[\![\{\mathfrak{B}/\alpha\}\mathfrak{A}]\!]_\sigma = [\![\mathfrak{A}]\!]_{\sigma,\alpha\mapsto[\mathfrak{B}]_\sigma}$.
Also note that $[\![\mathfrak{A}\cap\mathfrak{B}]\!]_\sigma = [\![\mathfrak{A}]\!]_\sigma \cap [\![\mathfrak{B}]\!]_\sigma$ and $[\![\forall\alpha\mathfrak{A}]\!]_\sigma = \bigcap_{Y\subseteq\mathcal{D}^*} [\![\mathfrak{A}]\!]_{\sigma,\alpha\mapsto Y}$.

An orthogonality model is a sufficiently rich structure for Adequacy to hold:

▶ **Lemma 29** (Adequacy Lemma). *If $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, then for all valuations σ and for all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]_\rho \in [\![\mathfrak{A}]\!]_\sigma$.*

Proof. By induction on $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, using axioms (A1), (A2) and (A3). See [7]. ◀

4.2 The special case of applicative structures

In the next section we present instances of orthogonality models. They will have in common that $\mathcal{E}$ is an applicative structure, as we have seen with I-filters. This motivates the following notion:

▶ **Definition 30** (Applicative orthogonality model).
An *applicative orthogonality model* is a 4-tuple $(\mathcal{E}, \mathcal{D}, @, [\![\_]\!]\_)$ where:

- $\mathcal{E}$ is a set, $\mathcal{D}$ is a subset of $\mathcal{E}$, @ is a (total) function from $\mathcal{E} \times \mathcal{E}$ to $\mathcal{E}$, and $[\![\_]\!]\_$ is a function (parameterised by an environment) from λ-terms to the support.
- $(\mathcal{E}, \mathcal{D}, \perp\!\!\!\perp, [\![\_]\!]\_)$ is an orthogonality model,
  where the relation $u \perp\!\!\!\perp \overrightarrow{v}$ is defined as $(u@\overrightarrow{v}) \in \mathcal{D}$
  (writing $u@\overrightarrow{v}$ for $(\dots(u@v_1)@\dots@v_n)$ if $\overrightarrow{v} = v_1 :: \dots v_n :: []$).

▶ Remark. Axioms (A1) and (A2) are ensured provided that $[\![M\ N]\!]_\rho = [\![M]\!]_\rho @ [\![N]\!]_\rho$ and $[\![x]\!]_\rho = \rho(x)$. These conditions can hold by definition (as in term models, cf. the next Section), or can be proved (as in Theorem 18, which also proves (A3)).

5 Three instances of orthogonality models

We now give three instances of (applicative) orthogonality models with well-chosen sets of values, applications, and interpretations of terms, so that, from $[\![M]\!] \in [\![A]\!]$, we can eventually derive the strong normalisation of the λ-term M.

The first two instances are term models: Terms are interpreted as themselves (see Definition 31), so the support is the set of all λ-terms seen as an applicative structure (using term application: $M_1 @^{\text{TERM}} M_2 := M_1\ M_2$).

▶ **Definition 31** (Interpretation of terms in a term model).

$$\begin{array}{lcl} [\![x]\!]_\rho^{\text{TERM}} & := & \rho(x) \\ [\![M_1\ M_2]\!]_\rho^{\text{TERM}} & := & [\![M_1]\!]_\rho^{\text{TERM}}\ [\![M_2]\!]_\rho^{\text{TERM}} \\ [\![\lambda x.M]\!]_\rho^{\text{TERM}} & := & \lambda x.[\![M]\!]_{\rho, x\mapsto x}^{\text{TERM}} \end{array}$$

▶ Remark. $[\![\{N/x\} M]\!]_\rho^{\text{TERM}} = [\![M]\!]_{\rho, x\mapsto [\![N]\!]_\rho^{\text{TERM}}}^{\text{TERM}}$

In the first instance, values are strongly normalising terms themselves; that instance rephrases, with an orthogonality model, standard proofs of strong normalisation by orthogonality or reducibility candidates [28, 24]. In the second instance, values are the terms that can be typed with intersection types, for instance in system $\lambda_\cap$.

The third instance is not a term model but a syntax-free model, where a term is interpreted as the filter of the intersection types that it can be assigned (e.g. in $\lambda_\cap$, see Definition 17), and orthogonality is defined in terms of filters being non-empty.

In the second and third instances, Strong Normalisation follows from the fact that terms typable with intersection types are themselves strongly normalising (Theorem 13 for $\lambda_\cap$).

▶ **Theorem 32.** *The structures*

- $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}} = (\Lambda, \mathsf{SN}, @^{\text{TERM}}, [\![\_]\!]^{\text{TERM}})$
- $\mathcal{M}^{\perp\!\!\!\perp}_{\cap} = (\Lambda, \Lambda_\cap, @^{\text{TERM}}, [\![\_]\!]^{\text{TERM}}_{\_})$ *(where $\Lambda_\cap$ is the set of λ-terms typable in $\lambda_\cap$)*
- $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}} = (\mathcal{E}, \mathcal{D}, @, [\![\_]\!]\_)$ *(with the four components as defined in Section 3)*

are applicative orthogonality models.

Indeed, as mentioned in Section 4.2, the applicative structures $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ already satisfy axioms (A1) and (A2) because of Definition 31. The structure $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ satisfies them, as well as axiom (A3), as immediate consequences of Theorem 18. Axiom (A3) holds in $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ because of their respective expansion properties:

▶ **Lemma 33** (Expansion).

1. *If $\{P/x\} M\ \overrightarrow{N} \in \mathsf{SN}$ and $P \in \mathsf{SN}$ then $(\lambda x.M)\ P\ \overrightarrow{N} \in \mathsf{SN}$.*
2. *If $\{P/x\} M\ \overrightarrow{N} \in \Lambda_\cap$ and $P \in \Lambda_\cap$ then $(\lambda x.M)\ P\ \overrightarrow{N} \in \Lambda_\cap$.*

Admittedly, once $\lambda_\cap$ has been proved to characterise SN (Theorems 13 and 39), the two points are identical and so are the two models $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$. But involving the bridge of

this characterisation goes much beyond than needed for either point: point 1 is a known fact of the literature; point 2 is a simple instance of Subject Expansion (Theorem 38, Appendix A) not requiring Subject Reduction (Theorem 12) while both are involved at some point in the more advanced property $\mathsf{SN} = \Lambda_\cap$. In brief, as we are interested in comparing proof *techniques* for the strong normalisation of System $\mathcal{S}$, the question of which properties are used and in which order matters.

▶ Remark. In both structures $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ we can check that:
For all $\overrightarrow{N} \in \mathsf{SN}^*$ and any term variable x, $x \perp\!\!\!\perp \overrightarrow{N}$.
Hence, for all valuations σ and all types $\mathfrak{A}$, $x \in [\![\mathfrak{A}]\!]_\sigma$.
For $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ we have instead: For all list of values $\overrightarrow{v}$, $\top \perp\!\!\!\perp \overrightarrow{v}$.
Hence, for all valuations σ and all types $\mathfrak{A}$, $\top \in [\![\mathfrak{A}]\!]_\sigma$.

Now using the Adequacy Lemma (Lemma 29), we finally get:

▶ **Corollary 34.** *If $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, then:*

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ *For all valuations σ and all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]^{\mathrm{TERM}}_\rho \in \mathsf{SN}$.*
Hence, $M \in \mathsf{SN}$.

$\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ *For all valuations σ and all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$*
there exist Γ and A such that $\Gamma \vdash_\cap [\![M]\!]^{\mathrm{TERM}}_\rho : A$. Hence, $M \in \mathsf{SN}$.

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ *For all valuations σ and all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]_\rho \neq \bot$.*
Hence, there exist Γ and A such that $\Gamma \vdash_\cap M : A$, and finally $M \in \mathsf{SN}$.

Proof.

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ The second statement is obtained by choosing σ to map every type variable to the empty set, and ρ to map every term variable to itself.

$\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ The second statement is obtained by choosing σ to map every type variable to the empty set, and ρ to map every term variable to itself; then we conclude using Theorem 13.

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ The first statement holds because $\bot \notin \mathcal{D}$ and $[\![\mathfrak{A}]\!]_\sigma \subseteq \mathcal{D}$. To prove the second, we need to show that there exist such a σ and such a ρ; take σ to map every type variable to the empty set and take ρ to map every term variable to $\top$. The final result comes from Theorem 13.

◀

6 A new light on polymorphism

System $\lambda_\cap$ is a convenient call on the journey from typability in $\mathcal{S}$ to strong normalisation, as it divides the path into two parts that are different in nature: first, proving that being typable in some system implies being typable in another (a result on the transformation of typing trees); second, proving that being typable in the latter system implies strong normalisation, which is trivial in the case of $\lambda_\cap$ (for each reduction makes the typing tree smaller).

The first part of the journey is described by the Adequacy Lemma in the filter models $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ and $\mathcal{M}^{\mathrm{i}}_{\mathcal{F}}$, as well as in the term model $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$: It shows that being typable in System $\mathcal{S}$ implies being typable in System $\lambda_\cap$. This is interesting in itself when applied to System F, as it sheds an interesting light on polymorphism:

As mentioned in the introduction, terms that are typed in System F, i.e. with *infinite* polymorphism, are strongly normalising (cf. model $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$), and can therefore be typed with intersection types (see e.g. Theorem 39), i.e. with *finite* polymorphism.

The cut or detour in the above proof can be eliminated, and this paper shows the resulting proof as the construction of another orthogonality model: namely, the term-model $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$,

where no mention is made of the strong normalisation property. The model construction is purely about the transformation of typing trees and appears to disregard normalisation properties. Yet it makes no explicit connection between the type of a λ-term in System F and the type that it gets in $\lambda_\cap$ via this method. The filter models $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ address this issue. Indeed, a System F type $\mathfrak{A}$ (say a closed one) is interpreted as a collection $[\![\mathfrak{A}]\!]$ of filters of intersection types; computing this can be done before inhabitants of types are even considered. Each of the filters in the collection represents the exact set of intersection types that a λ-term of type $\mathfrak{A}$ (in System F) could potentially get in $\lambda_\cap$.

▶ **Example 35.** For instance in $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$,

1. $\forall\alpha\alpha$ is interpreted as $[\![\forall\alpha\alpha]\!] = \bigcap_{X\in TP(\mathcal{D})} X = \{\top\}$.
   This means that a λ-term of type $\forall\alpha\alpha$ can necessarily be given any intersection type ($\top$ is the set of all intersection types).
2. $(\forall\alpha\alpha)\to(\forall\beta\beta)$ is interpreted as

$$\{\top\}\to\{\top\} = \{u\in\mathcal{D} \mid u@\top = \top\} = \{u\in\mathcal{D} \mid \forall F, \exists A, A\to F\in u\}$$

   Such a filter u contains an arrow towards each F-type.
3. $\forall\alpha\alpha\to\alpha$ is interpreted as $\{u\in\mathcal{D} \mid \forall X\in TP(\mathcal{D}), \forall v\in X, u@v\in X\}$.
   Such a filter u contains, for all F-types F, a type of the form $(F\cap\ldots\cap F)\to F$ (take $v =< F >$ and $X = \{v,\top\}$).

In brief, this interpretation of System F types transforms infinite polymorphism into finite polymorphism, generating collections of potential instances. The Adequacy Lemma then proves that those instances are sufficient to type the λ-terms.

In [32], a preorder with greatest lower bounds is identified in the syntax of System F types, into which intersection types can therefore be embedded. The converse is studied by a form of surjectivity of that embedding, but the example of $\vdash_F \lambda x.x\ \ldots\ x : (\forall\alpha\alpha)\to(\forall\beta\beta)$ is used to show that no intersection type exists that can type all the λ-terms of type $(\forall\alpha\alpha)\to(\forall\beta\beta)$. As we interpret System F types by *collections* of filters, different filters can be used for different terms.

While further work should relate our filters to inverse forms of the aforementioned embedding, the comparison with [32] also raises the question of whether or not the different versions of the proof that typable in System $\mathcal{S}$ implies typable in System $\lambda_\cap$, avoid the computation of the λ-terms down to their normal forms. This computation is of course present in the proof with the cut (the construction of model $\mathcal{M}^{\Vdash}_{\mathsf{SN}}$ combined with Theorem 39) and also present in [32] (type-preservation is proved by an induction on longest β-reduction sequences). One could argue that this computation is still present (yet hidden) in the construction of model $\mathcal{M}^{\Vdash}_{\cap}$ as it relies on the Subject Expansion property for $\lambda_\cap$ (Theorem 38).[4] It seems it is **not** the case for $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$: Indeed, while Adequacy does rely on the property that $[\![M]\!]_{\rho,x\mapsto[\![N]\!]_\rho}@\overrightarrow{v}\neq\bot$ implies $[\![(\lambda x.M)\ N]\!]_\rho@\overrightarrow{v}\neq\bot$, this property is **not** obtained by computing $\left\{{}^{N}\!/_{x}\right\}M$ but rather by analysing the typing trees of M and N separately.[5] This contrasts with the term models $\mathcal{M}^{\Vdash}_{\mathsf{SN}}$ and $\mathcal{M}^{\Vdash}_{\cap}$, where $\left\{{}^{N}\!/_{x}\right\}M$ is computed in the process and can generate new redexes to be further analysed.

Of course one could argue that, the typing trees in $\lambda_\cap$ of a λ-term M being bigger than the longest β-reduction sequence starting from M, producing one of them (be it with the

[4] It probably uses it as many times as there are reduction steps from a term to its normal form.
[5] The property $[\![M]\!]_{\rho,x\mapsto[\![N]\!]_\rho} = [\![\left\{{}^{N}\!/_{x}\right\}M]\!]_\rho$ holds, but not by definition.

Adequacy Lemma) is at least as hard as computing the term to its normal form. Yet $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ could well have been defined with a traditional idempotent intersection type system [9] (in which typing trees are not correlated to the lengths of β-reduction sequences), adapting directly [10] to System F.[6]

7 Conclusion

We have seen how the use of non-idempotent intersection types simplifies the methodology from [10] by cutting a second use of reducibility techniques to prove strong normalisation. We have seen how the corresponding filter model construction can be done by orthogonality techniques, with an abstract notion of orthogonality model. As illustrated in Section 5, this notion allows a lot of work (e.g. proving the Adequacy Lemma) to be factorised, while building models like $\mathcal{M}^{\Vdash}_{\mathsf{SN}}$, $\mathcal{M}^{\Vdash}_{\cap}$ and $\mathcal{M}^{\Vdash}_{\mathcal{F}}$. Note that, while $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ share the same ingredients $\mathcal{E}$, $\mathcal{D}$, @ and $[\![\_]\!]\_$, they are different in the way types are interpreted; see the discussion in [7].

We have also compared the models in the way they enlighten the transformation of infinite polymorphism into finite polymorphism, although more examples should be computed to illustrate and better understand the theoretical result. An objective could be to identify (and eliminate), in the interpretation of a type from System F, those filters that are not the interpretation of any term of that type. What could help this, is to force filters to be stable under type instantiation, in the view that interpretations of terms are generated by a single F-type, i.e. a *principal* type.

Proving Subject Expansion as generally as possible led us to identify a sub-reduction of β which also helps understanding how and when the semantics $[\![\_]\!]\_$ of terms is preserved, see Appendix A. This is similar to [1], and future work should adapt their methodology to accommodate our non-idempotent intersections.

Finally, as non-idempotent intersection types were used in [6] to measure the complexity of strongly normalising term, we would like to see whether we can adapt our model constructions to lift such results to the source typing systems. The hope would be to recover for instance results that are known for the simply-typed calculus [29, 5], with a methodology that can be adapted to other source systems such as System F.

References

1 Fabio Alessi, Franco Barbanera, and Mariangiola Dezani-Ciancaglini. Intersection types and lambda models. *Theoret. Comput. Sci.*, 355(2):108–126, 2006.

2 Patrick Baillot. Checking polynomial time complexity with types. In Ricardo A. Baeza-Yates, Ugo Montanari, and Nicola Santoro, editors, *IFIP TCS*, volume 223 of *IFIP Conference Proceedings*, pages 370–382. Kluwer, August 2002.

3 Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: a language for polynomial time computation. *CoRR*, cs.LO/0312015, 2003.

4 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. of Symbolic Logic*, 48(4):931–940, 1983.

5 Arnold Beckmann. Exact bounds for lengths of reductions in typed lambda-calculus. *J. of Symbolic Logic*, 66(3):1277–1285, 2001.

6 Alexis Bernadet and Stéphane Lengrand. Complexity of strongly normalising λ-terms via non-idempotent intersection types. In Martin Hofmann, editor, *Proc. of the 14th Int. Conf.*

[6] Non-idempotency is never used (but for the fact that it directly implies strong normalisation).

on Foundations of Software Science and Computation Structures (FOSSACS'11), volume 6604 of *LNCS*. Springer-Verlag, March 2011.

7 Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types, orthogonality and filter models - long version. Technical report, LIX, CNRS-INRIA-Ecole Polytechnique, June 2011.

8 Gérard Boudol, Pierre-Louis Curien, and Carolina Lavatelli. A semantics for lambda calculi with resources. *Math. Structures in Comput. Sci.*, 9(4):437–482, 1999.

9 M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19:139–156, 1978.

10 Thierry Coquand and Arnaud Spiwack. A proof of strong normalisation using domain theory. *Logic. Methods Comput. Science*, 3(4), 2007.

11 Vincent Danos and Jean-Louis Krivine. Disjunctive tautologies as synchronisation schemes. In Peter Clote and Helmut Schwichtenberg, editors, *Proc. of the 9th Annual Conf. of the European Association for Computer Science Logic (CSL'00)*, volume 1862 of *LNCS*, pages 292–301. Springer-Verlag, August 2000.

12 Daniel de Carvalho. Intersection types for light affine lambda calculus. *ENTCS*, 136:133–152, 2005.

13 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

14 M. Dezani-Ciancaglini, F. Honsell, and Y. Motohama. Compositional characterizations of lambda-terms using intersection types (extended abstract). In *MFCS: Symp. on Mathematical Foundations of Computer Science*, 2000.

15 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoret. Comput. Sci.*, 309(1-3):1–41, 2003.

16 Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for *lambda* -calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *Proc. of the 16th Annual Conf. of the European Association for Computer Science Logic (CSL'07)*, volume 4646 of *LNCS*, pages 253–267. Springer-Verlag, September 2007.

17 S. Ghilezan. Strong normalization and typability with intersection types. *Notre Dame J. Formal Loigc*, 37(1):44–52, 1996.

18 J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, 1972.

19 Jean-Yves Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–101, 1987.

20 A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Proc. of the 26th Annual ACM Symp. on Principles of Programming Languages (POPL'99)*, pages 161–174. ACM Press, 1999.

21 Jean-Louis Krivine. Typed lambda-calculus in classical Zermelo-Frænkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.

22 Yves Lafont. Soft linear logic and polynomial time. *Theoret. Comput. Sci.*, 318(1-2):163–180, 2004.

23 D. Leivant. Typing and computational properties of lambda expressions. *Theoret. Comput. Sci.*, 44(1):51–68, 1986.

24 Stéphane Lengrand and Alexandre Miquel. Classical F_ω, orthogonality and symmetric candidates. *Ann. Pure Appl. Logic*, 153:3–20, March 2008.

25 P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In Prakash Panangaden, editor, *20th Annual IEEE Symp. on Logic in Computer Science*, pages 82–91. IEEE Computer Society Press, June 2005.

26 Guillaume Munch-Maccagnoni. Focalisation and classical realisability. In Erich Grädel and Reinhard Kahle, editors, *Proc. of the 18th Annual Conf. of the European Association for*

Computer Science Logic (CSL'09), volume 5771 of *LNCS*, pages 409–423. Springer-Verlag, September 2009.

27 Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In Chris Okasaki and Kathleen Fisher, editors, *Proc. of the ACM Intern. Conf. on Functional Programming*, pages 138–149. ACM Press, September 2004.

28 Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. of Symbolic Logic*, 62(4):1461–1479, 1997.

29 Helmut Schwichtenberg. Complexity of normalization in the pure typed lambda calculus. In A. S. Troelstra and D. Van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, Amsterdam, 1982. North-Holland.

30 W. W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer-Verlag, 1975.

31 S. van Bakel. Intersection Type Assignment Systems. *Theoret. Comput. Sci.*, 151(2):385–435, 1995.

32 Hirofumi Yokouchi. Embedding a second order type system into an intersection type system. *Inform. and Comput.*, 117(2):206–220, 1995.

A Completeness and preservation of semantics

We obtain completeness by identifying a reduction strategy $\hookrightarrow$ (if a term can be β-reduced then one of its sub-terms can be reduced by $\hookrightarrow$). Proofs can be found in [7].

▶ **Definition 36** ($\hookrightarrow$)**.** We define the reduction $M \hookrightarrow_\Omega M'$, where Ω is either a term or ϵ (a dummy placeholder for which $fv(\epsilon) = \emptyset$) as follows:

$$\frac{x \in fv(M)}{(\lambda x.M)N \hookrightarrow_\epsilon \left\{{}^{N}\!/_{x}\right\}M} \qquad \frac{x \notin fv(M)}{(\lambda x.M)N \hookrightarrow_N M}$$

$$\frac{M_1 \hookrightarrow_\Omega M_1'}{M_1M_2 \hookrightarrow_\Omega M_1'M_2} \qquad \frac{M_2 \hookrightarrow_\Omega M_2'}{M_1M_2 \hookrightarrow_\Omega M_1M_2'} \qquad \frac{M \hookrightarrow_\Omega M' \quad x \notin fv(\Omega)}{\lambda x.M \hookrightarrow_\Omega \lambda x.M'}$$

▶ **Lemma 37** (Typing substitutions)**.** *If* $\Gamma \vdash_\cap \left\{{}^{N}\!/_{x}\right\}M : A$ *then there exists* Γ_1, Γ_2 *and* U *such that* $\Gamma \approx \Gamma_1 \cap \Gamma_2$, $\Gamma_1, x{:}U \vdash_\cap M{:}A$ *and* $\Gamma_2 \vdash_\cap N{:}U$.

▶ **Theorem 38** (Subject Expansion)**.** *Assume* $\Gamma' \vdash_\cap M'{:}A$ *and* $M \hookrightarrow_\Omega M'$.
Assume $\Delta \vdash_\cap N{:}B$ *if* $\Omega = N$, *otherwise let* $\Delta = ()$ *when* $\Omega = \epsilon$.
Then there exists Γ *such that* $\Gamma \approx \Gamma' \cap \Delta$ *and* $\Gamma \vdash_\cap M{:}A$.

Proof. First by induction on the derivation $M \hookrightarrow_\Omega M'$, then by induction on A, using Lemma 37 for the base case. ◀

▶ **Theorem 39** (Completeness)**.** *If* $M \in \mathsf{SN}$ *there exist* Γ *and* A *such that* $\Gamma \vdash_\cap M{:}A$.

▶ **Corollary 40.** *If* $M \hookrightarrow_\Omega M'$ *and* $M' \in \mathsf{SN}$ *and* $\Omega \in \mathsf{SN} \cup \{\epsilon\}$ *then* $M \in \mathsf{SN}$.

Preservation of term interpretation under reduction can also be described in terms of $\hookrightarrow$:

▶ **Theorem 41.** *1. If* $M \longrightarrow_\beta M'$ *then for all* ρ, $[\![M]\!]_\rho \subseteq [\![M']\!]_\rho$.
2. If $M \hookrightarrow_\epsilon M'$ *then* $[\![M]\!]_\rho = [\![M']\!]_\rho$.
3. If $M \hookrightarrow_N M'$ *and* $[\![N]\!]_\rho \neq \bot$, *then* $[\![M]\!]_\rho = [\![M']\!]_\rho$.
4. If $M \hookrightarrow_N M'$ *and* $[\![N]\!]_\rho = \bot$, *then* $[\![M]\!]_\rho = \bot$.
But there are cases where $M \longrightarrow_\beta M'$ and $[\![M]\!]_\rho \neq [\![M']\!]_\rho$ (even with $[\![M]\!]_\rho \neq \bot$).
Take $M := (\lambda z.(\lambda y.a)(zz))$ and $M' := \lambda z.a$.