ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

Annals of Pure and Applied Logic ▮ (▮▮▮▮) ▮▮▮–▮▮▮

ANNALS OF PURE AND APPLIED LOGIC

www.elsevier.com/locate/apal

Classical F_ω, orthogonality and symmetric candidates

Stéphane Lengrand[a,b,*], Alexandre Miquel[a]

[a] *PPS & Université Paris 7, 175 rue du Chevaleret, 75013 Paris, France*

[b] *School of Computer Science, University of St Andrews, North Haugh, St Andrews, Fife, KY16 9SX, Scotland, United Kingdom*

Abstract

We present a version of system F_ω, called F_ω^{c}, in which the layer of type constructors is essentially the traditional one of F_ω, whereas provability of types is classical. The proof-term calculus accounting for the classical reasoning is a variant of Barbanera and Berardi's symmetric λ-calculus.

We prove that the whole calculus is strongly normalising. For the layer of type constructors, we use Tait and Girard's reducibility method combined with orthogonality techniques. For the (classical) layer of terms, we use Barbanera and Berardi's method based on a symmetric notion of reducibility candidate. We prove that orthogonality does not capture the fixpoint construction of symmetric candidates.

We establish the consistency of F_ω^{c}, and relate the calculus to the traditional system F_ω, also when the latter is extended with axioms for classical logic.

Keywords: Classical logic; Classical version of system F_ω

1. Introduction

Approaches to a Curry–Howard correspondence for classical logic seem to converge towards the idea of programs equipped with some notion of control [18,4,22,20,8]. The general notion of reduction/computation is non-confluent but there are possible ways to restrict reductions and thus recover confluence.[1]

It is then tempting to try and build, on such a correspondence for classical logic, powerful type theories, such as those developed in intuitionistic logic (Pure Type Systems [2,3], Martin-Löf type theories [16]). Approaches to this task (in natural deduction) can be found in [21], in a framework *à la* Martin-Löf, and in [6] (but with a confluent restriction of the reductions of classical logic).

Intuitionistic type theories, however, exploit the fact that predicates are pure functions, which, when fully applied, give rise to formulae with logical meanings. The Curry–Howard correspondence in intuitionistic logic can then

* Corresponding author at: PPS & Université Paris 7, 175 rue du Chevaleret, 75013 Paris, France.
E-mail address: Lengrand@LIX.Polytechnique.fr (S. Lengrand).

[1] Two such canonical ways are related to CBV and CBN, with associated semantics given by CPS-translations, which correspond to the usual encodings of classical logic into intuitionistic logic known as "not–not"-translations.

0168-0072/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2008.01.005

describe these pure functions as the inhabitants of implicative types in a higher type layer (often called the layer of kinds).

On the other hand, inhabitants of implicative types in classical logic can be much wilder than pure functions (owing to the aforementioned notion of control), so it is not clear what meaning could be given to those simili-predicates, built from classical inhabitants of implicative types, and whose reductions may not even be confluent. However, such an issue is problematic only in the layer of types, a.k.a the *upper layer*, which various type theories "cleanly" separate from the layer of terms, a.k.a the *lower layer*.

This paper, which extends [15], shows that it is perfectly safe to have cohabiting layers with different logics, provided that the layer of types is free from any dependency on terms, i.e. that the system has no dependent types. For that we chose to tackle System F_ω [13]. We present here a version of it called F^{c}_ω that is classical in the following sense:

The upper layer is purely functional, i.e. intuitionistic: it is in fact the lambda-calculus extended with constants for logical connectives. Then, for those objects of the layer that are *types* (a.k.a. *formulae*), we have a notion of provability with proof derivations and proof-terms in the lower layer, which is here classical instead of intuitionistic.

The motivation for the choice of tackling F_ω is threefold:

- System F_ω is indeed the most powerful corner of Barendregt's Cube without dependent types [2,3].
- System F and the simply typed λ-calculus also cleanly separate the lower layer from the upper layer, but the latter is trivial as no computation happens there, in contrast to System F_ω which features computation in both layers, both strongly normalising.
- The version F^{c}_ω with a classical lower layer, in contrast to the intuitionistic one, features two *different* notions of computation (one intuitionistic and confluent, the other one classical and non-confluent), also both strongly normalising. Hence, F^{c}_ω represents an excellent opportunity to express and compare two techniques to prove strong normalisation that are based on the method of reducibility of Tait and Girard [13] and that look very similar, and solve a conjecture raised in [15] about one technique not capturing the other.

The strong normalisation of the upper layer (Section 3.1) represents an opportunity to rephrase the reducibility method [13] with the concepts and terminology of *orthogonality*, which provides a high level of abstraction and potential for modularity, but has a sparse literature (which includes [17]).

The technique for the strong normalisation of the lower layer (Section 3.2) adapts Barbanera and Berardi's method based on a symmetric notion of reducibility candidate [4] and a fixpoint construction. Previous works (e.g. [19,9]) adapt it to prove the strong normalisation of various sequent calculi, but (to our knowledge) not pushing it to such a typing system as that of F^{c}_ω (with a notion of computation on types). Note that we also introduce the notion of orthogonality in the proof technique (to elegantly express it and compare it to the proof for the upper layer).

The method works in fact without any surprise. Difficulties would come with dependent types (the only feature of Barendregt's Cube missing here), precisely because they would pollute the layer of types with non-confluence and unclear semantics.

The main purpose of presenting together the two proof techniques described above is in fact to express them whilst pointing out similarities, and to examine whether or not the concepts of the symmetric candidates method can be captured by the concept of orthogonality. In this paper we solve the conjecture of [15] by proving that it cannot.

Finally we prove the consistency of F^{c}_ω, and establish a formal connection with the traditional system F_ω, also when the latter uses extra axioms to allow classical reasoning.

Section 2 introduces F^{c}_ω. Section 3 establishes the strong normalisation of the layer of types, and that of the layer of terms. Section 4 compares the two proofs and solves the conjecture of [15]. Section 5 establishes some logical properties of F_ω such as consistency.

2. Syntax, reduction and typing of F^{c}_ω

2.1. Syntax

F^{c}_ω distinguishes four syntactic categories: *kinds*, *type constructors* (or *constructors* for short), *terms* and *programs*:

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Kinds	K, K'	$::=$	$\star \mid K \to K'$
Constructors	$A, B, C, \ldots$	$::=$	$\alpha \mid \alpha^\perp \mid \lambda\alpha : K \,.\, B \mid B\ A$
		$\mid$	$A \wedge B \mid A \vee B$
		$\mid$	$\forall\alpha : K \,.\, B \mid \exists\alpha : K \,.\, B$
Terms	$t, u, v, \ldots$	$::=$	$x \mid \mu x^A.p$
		$\mid$	$\langle t, u\rangle \mid \gamma x^A y^B.c$
		$\mid$	$\Lambda\alpha : K \,.\, t \mid \langle A, t\rangle$
Programs	p	$::=$	$\{t \mid u\}$

Kinds, that are exactly the same as in system F_ω [13,5], are a system of simple types for type constructors. (We use the word 'kind' to distinguish kinds from the types which appear at the level of type constructors.) The basic kind $\star$ is the kind of *types*, that is, the kind of all type constructors that represent types of terms—or *propositions/formulae* through the Curry–Howard correspondence.

Type constructors, often shortened as *constructors*, are basically simply-typed λ-terms with two binary operators $A \wedge B$ (conjunction), $A \vee B$ (disjunction) and two extra binders $\forall\alpha : K \,.\, A$ and $\exists\alpha : K \,.\, A$ to represent universal and existential quantification. (There is no primitive implication in the system.)

Following a presentation which is standard in linear logic [14], *negation* is a primitive construction only on variables, introducing a construction $\alpha^\perp$ for each variable α. The constructions $\forall\alpha : K \,.\, B$, $\exists\alpha : K \,.\, B$ and $\lambda\alpha : K \,.\, B$ then bind all free occurrences of the variable α in B, including those in subterms of the form $\alpha^\perp$. (In other words, the syntactic construction $\alpha^\perp$ is not a variable.) For instance, the type constructor

$$\neg = \lambda\alpha : \star \,.\, \alpha^\perp$$

is closed; this is the type constructor which represents negation as a function (of kind $\star \to \star$). Bound variables and α-conversion are treated as usual, and we sometimes omit the side-conditions avoiding variable capture when they can be easily recovered.

Negation is then *extended* as an involutive *operation* $A \mapsto A^\perp$ over the set of all constructors via de Morgan laws:

$$(\alpha)^\perp = \alpha^\perp \qquad (\alpha^\perp)^\perp = \alpha$$
$$(A \wedge B)^\perp = A^\perp \vee B^\perp \qquad (A \vee B)^\perp = A^\perp \wedge B^\perp$$
$$(\forall\alpha : K \,.\, B)^\perp = \exists\alpha : K \,.\, B^\perp \qquad (\exists\alpha : K \,.\, B)^\perp = \forall\alpha : K \,.\, B^\perp$$
$$(\lambda\alpha : K \,.\, B)^\perp = \lambda\alpha : K \,.\, B^\perp \qquad (B\ A)^\perp = B^\perp\ A.$$

Notice how negation propagates through λ-abstraction and application. In our calculus, the notation $A^\perp$ is not only meaningful for types (that is, constructors of kind $\star$), but it is defined for all type constructors. With negation extended to all type constructors we can define implication $A \Rightarrow B$ as $(A^\perp) \vee B$.

The computation rules of negation are incorporated into the calculus by extending the definition of the (external) operation of substitution, written $B\{\alpha \backslash A\}$, to the case where B is a negated variable, as shown in Fig. 1. (Notice that in the last three cases, the bound variable α can be appropriately renamed so that the side-condition $\beta \neq \alpha$ holds and variable capture is avoided.)

This (extended) notion of substitution satisfies the following properties:

Remark 1.

1. $(A\{\alpha \backslash B\})^\perp = A^\perp\{\alpha \backslash B\}$.
2. $A\{\alpha \backslash B\}\{\beta \backslash C\} = A\{\beta \backslash C\}\{\alpha \backslash B\{\beta \backslash C\}\}$.

The (proof-)terms of our calculus are basically the terms of Barbanera and Berardi's symmetric λ-calculus, with the difference that connectives are treated multiplicatively. In particular, disjunction is treated as a negative connective whose proofs are built using a double binder written $\gamma x^A y^B.p$. On the other hand, proofs of conjunction are introduced as usual, using the pairing construct written $\langle t, u\rangle$.

Finally, programs are built by making two terms t and u interact using a construction written $\{t \mid u\}$, where each term can be understood as the evaluation context of the other term. We assume that this construction is symmetric,

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

$\alpha\{\beta\backslash C\}$	$=$	α	$(\beta \neq \alpha)$
$\beta\{\beta\backslash C\}$	$=$	C	
$\alpha^{\perp}\{\beta\backslash C\}$	$=$	$\alpha^{\perp}$	$(\beta \neq \alpha)$
$\beta^{\perp}\{\beta\backslash C\}$	$=$	$C^{\perp}$	
$(A \wedge B)\{\beta\backslash C\}$	$=$	$A\{\beta\backslash C\} \wedge B\{\beta\backslash C\}$	
$(A \vee B)\{\beta\backslash C\}$	$=$	$A\{\beta\backslash C\} \vee B\{\beta\backslash C\}$	
$(B\ A)\{\beta\backslash C\}$	$=$	$B\{\beta\backslash C\}\ A\{\beta\backslash C\}$	
$(\lambda\alpha : K\ .\ A)\{\beta\backslash C\}$	$=$	$\lambda\alpha : K\ .\ A\{\beta\backslash C\}$	$(\beta \neq \alpha)$
$(\forall\alpha : K\ .\ A)\{\beta\backslash C\}$	$=$	$\forall\alpha : K\ .\ A\{\beta\backslash C\}$	$(\beta \neq \alpha)$
$(\exists\alpha : K\ .\ A)\{\beta\backslash C\}$	$=$	$\exists\alpha : K\ .\ A\{\beta\backslash C\}$	$(\beta \neq \alpha)$

Fig. 1. Substitution in the upper layer.

that is, that $\{t \mid u\}$ and $\{u \mid t\}$ denote the same program. Henceforth, terms and programs are considered up to this equality together with α-conversion.

2.2. Reduction and typing for types

The reduction relation on the layer of type constructors is β-reduction, which is defined as usual as the contextual closure of the relation

$$(\lambda\alpha : K\ .\ B)A \quad \longrightarrow_{\beta} \quad B\{\alpha\backslash A\}.$$

However, the extension of the definition of substitution to negated variables mechanically enhances β-reduction in such a way that we get de Morgan equalities for free:

$$\neg(A \wedge B) \ =_{\beta}\ \neg A \vee \neg B \qquad \neg(A \vee B) \ =_{\beta}\ \neg A \wedge \neg B$$
$$\neg(\forall\alpha : K\ .\ B) \ =_{\beta}\ \exists\alpha : K\ .\ \neg B \qquad \neg(\exists\alpha : K\ .\ B) \ =_{\beta}\ \forall\alpha : K\ .\ \neg B.$$

(Here, $\neg$ denotes the type constructor $\lambda\alpha : \star\ .\ \alpha^{\perp}$, and $=_{\beta}$ denotes the congruence generated by $\longrightarrow_{\beta}$.)

Lemma 2. — *If* $A \longrightarrow_{\beta} B$ *then* $A^{\perp} \longrightarrow_{\beta} B^{\perp}$.

Proof. This is a corollary of Remark 1.1. □

Proposition 3. — *The (enhanced)* β*-reduction on type constructors is confluent.*

Proof. This is proved by introducing the corresponding notion of parallel reduction, following Tait and Martin-Löf [1], and using Lemma 2. □

Typing contexts for variables of type constructors, that we call *signatures*, are consistent[2] finite sets of declarations of the form $(\alpha : K)$:

Signatures $\quad \Sigma \ ::= \ \alpha_1 : K_1, \ldots, \alpha_n : K_n.$

The inference rules of the typing judgement $\Sigma \vdash A : K$ ('In the signature Σ, A is a constructor of kind K') are given in Fig. 2.

The typing system satisfies the following properties:

Proposition 4. 1. *(Weakening) If* $\Sigma \vdash A : K$ *then* $\Sigma, \alpha : K' \vdash A : K$.
2. *(Negation preserves typing) If* $\Sigma \vdash A : K$ *then* $\Sigma \vdash A^{\perp} : K$.
3. *(Substitution is well-typed) If* $\Sigma \vdash A : K$ *and* $\Sigma, \alpha : K \vdash B : K'$ *then* $\Sigma \vdash B\{\alpha\backslash A\} : K'$.

It also satisfies Subject reduction:

Proposition 5 (*Subject Reduction*). — *If* $\Sigma \vdash A : K$ *and if* $A \longrightarrow_{\beta} A'$, *then* $\Sigma \vdash A' : K$.

[2] By *consistent* is meant that if $\alpha : K_1$ and $\alpha : K_2$ are in Σ, then $K_1 = K_2$.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_{ω}, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

$$\frac{}{\Sigma \vdash \alpha : K}\,(\alpha : K) \in \Sigma \qquad \frac{}{\Sigma \vdash \alpha^{\perp} : K}\,(\alpha : K) \in \Sigma$$

$$\frac{\Sigma, \alpha : K \vdash B : K'}{\Sigma \vdash \lambda\alpha : K \,.\, B : K \to K'} \qquad \frac{\Sigma \vdash B : K \to K' \qquad \Sigma \vdash A : K}{\Sigma \vdash B\ A : K'}$$

$$\frac{\Sigma \vdash A : \star \qquad \Sigma \vdash B : \star}{\Sigma \vdash A \wedge B : \star} \qquad \frac{\Sigma \vdash A : \star \qquad \Sigma \vdash B : \star}{\Sigma \vdash A \vee B : \star}$$

$$\frac{\Sigma, \alpha : K \vdash B : \star}{\Sigma \vdash \forall\alpha : K \,.\, B : \star} \qquad \frac{\Sigma, \alpha : K \vdash B : \star}{\Sigma \vdash \exists\alpha : K \,.\, B : \star}$$

Fig. 2. Typing rules for type constructors.

2.3. Reduction and typing for terms and programs

The reduction system of the lower layer of F^{c}_{ω}, presented in Fig. 3, applies on programs, but the contextual closure equip both programs and terms with a reduction relation. Note that the contextual closure also incorporates reduction of type constructors: for instance, $\Upsilon x^{A} y^{B}.t$ (β-)reduces to $\Upsilon x^{A'} y^{B}.t$ if $A \longrightarrow_{\beta} A'$. Finally, recall that the programs $\{t \mid u\}$ and $\{u \mid t\}$ are identified, so we consider the reduction relation *modulo* the congruence defined by this identity and we denote it $\longrightarrow_{F^{c}_{\omega}}$.

$$\begin{array}{lll}
\{\mu x^{A}.p \mid t\} & \longrightarrow_{\mu} & p\{x \backslash t\} \\
\{\langle t_1, t_2\rangle \mid \Upsilon x_1^{A} x_2^{B}.p\} & \longrightarrow_{\wedge\vee_l} & \{t_1 \mid \mu x_1^{A}.\{t_2 \mid \mu x_2^{B}.p\}\} \\
\text{or} & \longrightarrow_{\wedge\vee_r} & \{t_2 \mid \mu x_2^{B}.\{t_1 \mid \mu x_1^{A}.p\}\} \\
\{\Lambda\alpha : K \,.\, t \mid \langle A, u\rangle\} & \longrightarrow_{\forall\exists} & \{t\{\alpha \backslash A\} \mid u\}
\end{array}$$

Fig. 3. Reduction rules on terms and programs.

As in Barbanera and Berardi's symmetric λ-calculus [4] or in Curien and Herbelin's $\lambda\mu\tilde{\mu}$-calculus [8], the critical pair

$$\begin{array}{ccc}
 & \{\mu x^{A}.p \mid \mu y^{A'}.q\} & \\
 \swarrow & & \searrow \\
p\{x \backslash \mu y^{A'}.q\} & & q\{y \backslash \mu x^{A}.p\}
\end{array}$$

cannot be joined, and in fact reduction is not confluent in general in this layer (see Example 2 below).

Typing contexts for variables of terms, that we simply call *contexts*, are consistent[3] finite sets of declarations of the form $(x : A)$:

Contexts $\quad \Gamma \ ::= \ x_1 : A_1, \ldots, x_n : A_n.$

Since types A that appear in a context may depend on constructor variables, each context Γ only makes sense in a given signature Σ. In what follows, we say that a context Γ is *well-formed* in a signature Σ and write $\mathsf{wf}_{\Sigma}(\Gamma)$ if for all declarations $(x : A) \in \Gamma$, the judgement $\Sigma \vdash A : \star$ is derivable.

From this, we define two judgements, namely:

$\Gamma \vdash_{\Sigma} t : A$ 'In the signature Σ and context Γ, the term t has type A'
$\Gamma \vdash_{\Sigma} p \diamond$ 'In the signature Σ and context Γ, the program p is well-formed'.

[3] By *consistent* is meant that if $x : A_1$ and $x : A_2$ are in Γ, then $A_1 = A_2$.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_{ω}, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Both judgements are defined by mutual induction from the rules given in Fig. 4.

$$\frac{\mathsf{wf}_\Sigma(\Gamma)}{\Gamma \vdash_\Sigma x : A}\ (x : A) \in \Gamma \qquad \frac{\Gamma, x : A \vdash_\Sigma p \diamond}{\Gamma \vdash_\Sigma \mu x^A.p : A^\perp}$$

$$\frac{\Gamma \vdash_\Sigma t : A \qquad \Gamma \vdash_\Sigma u : B}{\Gamma \vdash_\Sigma \langle t, u\rangle : A \wedge B} \qquad \frac{\Gamma, x : A, y : B \vdash_\Sigma p \diamond}{\Gamma \vdash_\Sigma \gamma x^A y^B.p : A^\perp \vee B^\perp}$$

$$\frac{\Gamma \vdash_{\Sigma,\alpha:K} t : B}{\Gamma \vdash_\Sigma \Lambda\alpha : K\,.\,t : \forall\alpha : K\,.\,B} \qquad \frac{\Sigma \vdash A : K \qquad \Gamma \vdash_\Sigma u : B\{\alpha\backslash A\}}{\Gamma \vdash_\Sigma \langle A, u\rangle : \exists\alpha : K\,.\,B}$$

$$\frac{\Gamma \vdash_\Sigma t : A \qquad \Sigma \vdash A' : \star}{\Gamma \vdash_\Sigma t : A'}\ A =_\beta A' \qquad \frac{\Gamma \vdash_\Sigma t : A \qquad \Gamma \vdash_\Sigma u : A^\perp}{\Gamma \vdash_\Sigma \{t \mid u\} \diamond}$$

Fig. 4. Typing rules for terms and programs.

This typing system satisfies the following properties:

Proposition 6. 1. *(Weakening of signature) If* $\Gamma \vdash_\Sigma t : B$ *(resp.* $\Gamma \vdash_\Sigma p \diamond$*) then* $\Gamma \vdash_{\Sigma,\alpha:K} t : B$ *(resp.* $\Gamma \vdash_{\Sigma,\alpha:K} p \diamond$*.)*
2. *(Weakening of context) If* $\Gamma \vdash_\Sigma t : B$ *(resp.* $\Gamma \vdash_\Sigma p \diamond$*) and* $\Sigma \vdash A : K$ *then* $\Gamma, x : A \vdash_\Sigma t : B$ *(resp.* $\Gamma, x : A \vdash_\Sigma p \diamond$*.)*
3. *(Substitution of constructors is well-typed) If* $\Sigma \vdash A : K$ *and* $\Gamma \vdash_{\Sigma,\alpha:K} t : B$ *(resp.* $\Gamma \vdash_{\Sigma,\alpha:K} p \diamond$*) then* $\Gamma\{\alpha\backslash A\} \vdash_\Sigma t\{\alpha\backslash A\} : B\{\alpha\backslash A\}$ *(resp.* $\Gamma\{\alpha\backslash A\} \vdash_\Sigma p\{\alpha\backslash A\} \diamond$*).*
4. *(Substitution of terms is well-typed) If* $\Gamma \vdash_\Sigma u : A$ *and* $\Gamma, x : A \vdash_\Sigma t : B$ *(resp.* $\Gamma, x : A \vdash_\Sigma p \diamond$*) then* $\Gamma \vdash_\Sigma t\{x\backslash u\} : B$ *(resp.* $\Gamma \vdash_\Sigma p\{x\backslash u\} \diamond$*).*

And again it also satisfies Subject reduction, despite the non-deterministic nature of reduction:

Proposition 7 (*Subject-Reduction*).

1. *If* $\Gamma \vdash_\Sigma t : A$ *and* $t \longrightarrow_{F^c_\omega} t'$*, then* $\Gamma \vdash_\Sigma t' : A$*.*
2. *If* $\Gamma \vdash_\Sigma p \diamond$ *and* $p \longrightarrow_{F^c_\omega} p'$*, then* $\Gamma \vdash_\Sigma p' \diamond$*.*

Proof. By simultaneous induction on the judgements $\Gamma \vdash_\Sigma t : A$ and $\Gamma \vdash_\Sigma p \diamond$. □

Example 1. Here is a proof of the Law of excluded middle:

$$\dfrac{\dfrac{\dfrac{x : \alpha^\perp, y : \alpha \vdash_{\alpha:\star} x : \alpha^\perp \qquad x : \alpha^\perp, y : \alpha \vdash_{\alpha:\star} y : \alpha}{x : \alpha^\perp, y : \alpha \vdash_{\alpha:\star} \{x \mid y\} \diamond}}{\vdash_{\alpha:\star} \gamma x^{\alpha^\perp} y^\alpha.\{x \mid y\} : \alpha \vee (\alpha^\perp)}}{\vdash \Lambda\alpha : \star\,.\,\gamma x^{\alpha^\perp} y^\alpha.\{x \mid y\} : \forall\alpha : \star\,.\,\alpha \vee (\alpha^\perp)}\ .$$

Example 2. Here is Lafont's example of non-confluence. Suppose $\Gamma \vdash_{\alpha:\star} p_1 \diamond$ and $\Gamma \vdash_{\alpha:\star} p_2 \diamond$. With $x \notin \mathcal{FV}(p_1)$ and $y \notin \mathcal{FV}(p_2)$, by weakening we get

$$\dfrac{\dfrac{\Gamma, x : \alpha \vdash_{\alpha:\star} p_1 \diamond}{\Gamma \vdash_{\alpha:\star} \mu x^\alpha.p_1 : \alpha^\perp} \qquad \dfrac{\Gamma, y : \alpha^\perp \vdash_{\alpha:\star} p_2 \diamond}{\Gamma \vdash_{\alpha:\star} \mu y^{\alpha^\perp}.p_2 : \alpha}}{\Gamma \vdash_{\alpha:\star} \{\mu x^\alpha.p_1 \mid \mu y^{\alpha^\perp}.p_2\} \diamond}\ .$$

But $\{\mu x^\alpha.p_1 \mid \mu y^{\alpha^\perp}.p_2\} \longrightarrow^*_\mu p_1$ or $\{\mu x^\alpha.p_1 \mid \mu y^{\alpha^\perp}.p_2\} \longrightarrow^*_\mu p_2$. And unless the system is proof-irrelevant, p_1 and p_2 can be completely different.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Definition 1 (*Incestuous Pairs*). We call *incestuous pair* a program of one of the following forms:

PAIR–PAIR	$\{\langle t_1, u_1\rangle \mid \langle t_2, u_2\rangle\}$
LAMBDA–LAMBDA	$\{\Upsilon x_1^{A_1} y_1^{B_1}.p_1 \mid \Upsilon x_2^{A_2} y_2^{B_2}.p_2\}$
∀LAMBDA–∀LAMBDA	$\{\Lambda\alpha_1 : K\,.\,t_1 \mid \Lambda\alpha_2 : K\,.\,t_2\}$
∃WITNESS–∃WITNESS	$\{\langle A_1, t_1\rangle \mid \langle A_2, t_2\rangle\}$
LAMBDA–∀LAMBDA	$\{\Upsilon x_1^{A_1} y_1^{B_1}.p_1 \mid \Lambda\alpha : K\,.\,t_2\}$
PAIR–∀LAMBDA	$\{\langle t_1, u_1\rangle \mid \Lambda\alpha : K\,.\,t_2\}$
LAMBDA–∃WITNESS	$\{\Upsilon x_1^{A_1} y_1^{B_1}.p_1 \mid \langle A_2, t_2\rangle\}$
PAIR–∃WITNESS	$\{\langle t_1, u_1\rangle \mid \langle A_2, t_2\rangle\}$.

Proposition 8. — *Incestuous pairs can never be typed.*

Proof. The upper layer is confluent, so $A \wedge B \neq_\beta C \vee D$, $\forall\alpha : K\,.\,A \neq_\beta \exists\alpha' : K'\,.\,B$, $A \wedge B \neq_\beta \exists\alpha' : K'\,.\,B$, $\forall\alpha : K\,.\,A \neq_\beta C \vee D$, $A \wedge B \neq_\beta \forall\alpha : K\,.\,A$ and $\exists\alpha' : K'\,.\,B \neq_\beta \exists\alpha' : K'\,.\,B$. □

Finally, note that, in contrast to Barbanera and Berardi's symmetric λ-calculus, our design choices for the typing rules are such that, by constraining terms and programs to be linear, we get exactly the multiplicative fragment of linear logic [14].

3. Strong normalisation

In this section we prove the strong normalisation of the two layers of F_ω^{c}. In both cases the method is based on the reducibility technique of Tait and Girard [13].

This consists in building a strongly normalising model of the calculus, interpreting kinds (resp. types) as sets of strongly normalising type constructors (resp. pairs of strongly normalising terms). By definition, these sets (resp. pairs of sets) contain the basic constructs that introduce a connective (resp. that introduce dual connectives).

This is sufficient to treat most cases of the induction to prove the soundness theorem (which roughly states that being typed implies being in the model, hence being strongly normalising), but for the other cases we need the property that the interpretation of kinds (resp. types) is *saturated*, so we extend these interpretations by a completion process.

Now the completion process is precisely where the proofs of strong normalisation of the two layers differ: For the upper layer we simply use a completion by bi-orthogonality and this gives us the desired saturation property. For the lower layer, the completion process is obtained by Barbanera and Berardi's fixpoint construction. We discuss this difference in Section 4.

3.1. Strong normalisation of type constructors

In this section we prove that all well-typed constructors are strongly normalisable. For that, let us write SN_C the set of all strongly normalisable type constructors.

We call a *stack* (of type constructors) any finite sequence $S = (A_1, \ldots, A_n)$ of type constructors. Given a type constructor B and a stack $S = (A_1, \ldots, A_n)$, we define the application BS by setting $BS = BA_1 \cdots A_n$.

We say that a stack $S = (A_1, \ldots, A_n)$ is strongly normalisable when all its elements $A_1, \ldots, A_n$ are strongly normalisable. The set of all strongly normalisable stacks is written SN_C^*. In general, applying a strongly normalisable constructor $B \in \mathsf{SN}_\mathsf{C}$ to a strongly normalisable stack $S \in \mathsf{SN}_\mathsf{C}^*$ does not yield a strongly normalisable constructor BS. In the case where $BS \in \mathsf{SN}_\mathsf{C}$, we thus say that B and S are *orthogonal*, and write $B \perp S$.

Given a subset $X \subset \mathsf{SN}_\mathsf{C}$, we write $X^\perp$ the subset of SN_C^* called the *orthogonal of X* and defined by

$$X^\perp = \{S \in \mathsf{SN}_\mathsf{C}^* \mid B \perp S \text{ for all } B \in X\}.$$

Similarly, the orthogonal $Y^\perp \subset \mathsf{SN}_\mathsf{C}$ of a subset $Y \subset \mathsf{SN}_\mathsf{C}^*$ is defined as

$$Y^\perp = \{B \in \mathsf{SN}_\mathsf{C} \mid B \perp S \text{ for all } S \in Y\}.$$

The operation $X \mapsto X^\perp$ fulfils the usual properties of orthogonality on SN_C (as well as on SN_C^*):

1. $X \subseteq X'$ entails $X'^\perp \subseteq X^\perp$ (contravariance)

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

2. $X \subseteq X^{\perp\perp}$ (closure)
3. $X^{\perp\perp\perp} = X^{\perp}$ (tri-orthogonal).

Definition 2 (*Reducibility Candidate*). — We call a *reducibility candidate* any subset $X \subseteq \mathsf{SN_C}$ such that $X = X^{\perp\perp}$.

Notice that reducibility candidates are precisely the subsets $X \subseteq \mathsf{SN_C}$ of the form $X = Y^{\perp}$ for some subset $Y \subseteq \mathsf{SN_C^*}$. In particular, $\mathsf{SN_C}$ is a reducibility candidate, since $\mathsf{SN_C} = \{()\}^{\perp}$ (writing () for the empty stack).

Reducibility candidates enjoy the following properties:

Proposition 9. — *For all reducibility candidates X:*

1. $X \subset \mathit{SN_C}$;
2. *X contains all variables* α *and negated variables* $\alpha^{\perp}$;
3. *X is closed under* β*-reduction, that is:*
   if $B \in X$ *and* $B \longrightarrow_{\beta} B'$, *then* $B' \in X$;
4. *X is* saturated, *i.e. closed under head* β*-expansion:*
   if $B\{\alpha\backslash A\} \in X$ *and* $A \in \mathit{SN_C}$, *then* $(\lambda\alpha : K \,.\, B)A \in X$.

Proof. Item 1 holds by definition. Item 2 holds since αS (resp. $\alpha^{\perp} S$) is strongly normalisable as soon as the stack S is strongly normalisable. Item 3 holds since strongly normalisable type constructors are closed under β-reduction. Finally, item 4 is a consequence of the following property: If the type constructors A and $B\{\alpha\backslash A\}A_1 \cdots A_n$ are strongly normalisable, then so is $(\lambda\alpha : K \,.\, B)AA_1 \cdots A_n$. □

Definition 3 (*Set Constructions*). We define the following abbreviations:

$$X \to X' = \{B \in \mathsf{SN_C} \mid \forall A{\in}X,\ \ (BA){\in}X'\}$$
$$\lambda X \,.\, X' \ = \{\lambda\alpha : K \,.\, B \in \mathsf{SN_C} \mid \forall A \in X,\ \ B\{\alpha\backslash A\} \in X'\}.$$

Lemma 10. — *For all subsets* $X \subset \mathit{SN_C}$ *and* $Y \subset \mathit{SN_C^*}$,

$$X \to Y^{\perp} = (\lambda X \,.\, Y^{\perp})^{\perp\perp}.$$

Proof. Since $Y^{\perp}$ is a reducibility candidate ($Y^{\perp} = Y^{\perp\perp\perp}$), it is saturated, that is, if $B\{\alpha\backslash A\} \in Y^{\perp}$ then $(\lambda\alpha : K \,.\, B)\ A \in Y^{\perp}$. Hence, we get $\lambda X \,.\, Y^{\perp} \subseteq X \to Y^{\perp}$.

Now notice that $X \to Y^{\perp} = \{A :: S \mid A \in X, S \in Y\}^{\perp}$ (where $A :: S$ denotes the consing operation on stacks), so it is a reducibility candidate as well, and thus $(\lambda X \,.\, Y^{\perp})^{\perp\perp} \subseteq X \to Y^{\perp}$.

This direction is enough for the proof of strong normalisation, but the reverse direction can also be proved:

Assuming $C \in X \to Y^{\perp}$ and $S \in (\lambda X \,.\, Y^{\perp})^{\perp}$, we want to show $C \perp S$. Since $C \in \mathsf{SN_C}$ and $S \in \mathsf{SN_C^*}$, any infinite reduction sequence would start with:

$$C\ S \longrightarrow_{\beta}^{*} (\lambda\alpha : K \,.\, B)\ S'$$

with $S \longrightarrow_{\beta}^{*} S' \in (\lambda X \,.\, Y^{\perp})^{\perp}$ and $C \longrightarrow_{\beta}^{*} \lambda\alpha : K \,.\, B \in (X \to Y^{\perp})$, for which $\lambda\alpha : K \,.\, B \in \lambda X \,.\, Y^{\perp}$. □

From this, we interpret each kind K as a reducibility candidate:

Definition 4 (*Interpretation of Kinds*). The interpretation $[K]$ of a kind K is a reducibility candidate defined by induction on K as follows:

$$[\star] \qquad = \mathsf{SN_C}$$
$$[K \to K'] \ = [K] \to [K'] = (\lambda[K] \,.\, [K'])^{\perp\perp}.$$

Lemma 11. — *If the typing judgment* $\alpha_1 : K_1, \ldots, \alpha_n : K_n \vdash B : K$ *is derivable, then for all* $A_1 \in [K_1]$, …, $A_n \in [K_n]$ *one has*

$$B\{\alpha_1, \ldots, \alpha_n\backslash A_1, \ldots, A_n\} \in [K]$$

(where $B\{\alpha_1, \ldots, \alpha_n\backslash A_1, \ldots, A_n\}$ *denotes the parallel substitution of the type constructors* $A_1, \ldots, A_n$ *to the variables* $\alpha_1, \ldots, \alpha_n$ *in the type constructor B).*

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_{ω}, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Proof. By induction on the derivation of $\alpha_1 : K_1, \ldots, \alpha_n : K_n \vdash B : K$. □

From this we get:

Theorem 12. — *It $\Sigma \vdash B : K$, then B is strongly normalisable.*

Proof. Apply Lemma 11 with $A_1 = \alpha_1, \ldots, A_n = \alpha_n$ (identity substitution), using item 2 of Proposition 9. □

3.2. Strong normalisation of terms

This proof is adapted from those of [4,19,9] for the symmetric λ-calculus [4], the $\overline{\lambda}\mu\tilde{\mu}$-calculus [8], and the *dual calculus* [23] (which are based on a bi-sided sequent calculi), respectively. They all use Barbanera and Berardi's *symmetric candidates*, with a fixpoint construct to capture the non-confluence of classical logic.

As usual with the reducibility method we construct a model of the calculus by interpreting types (here, type constructors and type lists) as sets of terms. However, the second-order quantification that appears in System F or F_ω is conveniently interpreted as a set intersection only if terms do not display type annotations. We therefore start by defining such term and programs, i.e. Curry-style terms and programs:

$$\begin{array}{lrcl} \textbf{Curry-style terms} & t, u, v, \ldots & ::= & x \mid \mu x.p \mid \langle t, u\rangle \mid \Upsilon x\, y.p \mid \Lambda\_.t \mid \langle \_, t\rangle \\ \textbf{Curry-style programs} & p & ::= & \{t \mid u\}. \end{array}$$

The corresponding reduction rules that are shown in Fig. 5 define the Curry-style reduction $\longrightarrow_{F^{\mathsf{c}}_\omega}$ as well as the set SN of strongly normalising Curry-style terms and Curry-style programs. On the other hand, we write $\mathsf{SN}^{F^{\mathsf{c}}_\omega}$ to denote the set of all strongly normalising Church-style terms and programs.

$\{\mu x.p \mid t\}$	$\longrightarrow$	$p\{x\backslash t\}$
$\{\langle t_1, t_2\rangle \mid \Upsilon x_1 x_2.p\}$	$\longrightarrow$	$\{t_1 \mid \mu x_1.\{t_2 \mid \mu x_2.p\}\}$
	or	$\{t_2 \mid \mu x_2.\{t_1 \mid \mu x_1.p\}\}$
$\{\Lambda\_.t \mid \langle \_, u\rangle\}$	$\longrightarrow$	$\{t \mid u\}$

Fig. 5. Reductions without types.

Definition 5. — The type-erasure operation from terms (resp. programs) to Curry-style terms (resp. Curry-style programs) is recursively defined by:

$$\begin{array}{ll} \|x\| & = x \\ \|\langle t, u\rangle\| & = \langle \|t\|, \|u\|\rangle \\ \|\Upsilon x^A y^B.p\| & = \Upsilon x\, y.\|p\| \\ \|\mu x^A.p\| & = \mu x.\|p\| \\ \|\Lambda\alpha : K\,.\,t\| & = \Lambda\_.\|t\| \\ \|\langle A, t\rangle\| & = \langle \_, \|t\|\rangle \\ \|\{t \mid u\}\| & = \{\|t\| \mid \|u\|\}. \end{array}$$

Note that by erasing the types we still keep, in Curry-style programs, a trace of the constructs introducing the ∀ and ∃ quantifiers. Thus, it is slightly different from the traditional Curry-style polymorphism of system F or F_ω, but this trace turns out to be important in classical logic: if we removed it, we could make some μ–μ critical pair appear that was not present in the original program with type annotations, and one of the two reductions might not satisfy subject reduction.[4]

[4] This is a general problem of polymorphism and classical logic with non-confluent reduction: for instance the spirit of *intersection types* [7], which represent finite polymorphism, is to give several types to *the same program*, free from any trace of where the typing rules for intersection types have been used in its typing derivation. In that case again, non-confluent reductions of classical logic often fail to satisfy subject reduction.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Lemma 13. — *If all type constructors in a Church-style proof-term t are strongly normalising (for β) and if $\|t\| \in \mathsf{SN}$, then $t \in \mathsf{SN}^{F^c_\omega}$.*

Proof. Let $\mathcal{M}(t)$ be the multiset of all the type constructors appearing in t (easily defined by induction on t —e.g. $\mathcal{M}(\gamma x^A y^B.t) = \{\!\{A, B\}\!\} \cup \mathcal{M}(t)$). By assumption, all such type constructors are strongly normalising, so we can consider the standard multiset order based on the terminating β-reduction (on type constructors).
Every reduction from t decrease the pair $(\|t\|, \mathcal{M}(t))$ in lexicographic order. □

Definition 6 (*Orthogonality*). • We say that a Curry-style term t is *orthogonal* to a Curry-style term u, written $t \perp u$, if $\{t \mid u\} \in \mathsf{SN}$.
• We say that a set $\mathcal{U}$ of Curry-style terms is *orthogonal* to a set $\mathcal{V}$ of Curry-style terms, written $\mathcal{U} \perp \mathcal{V}$, if $\forall t \in \mathcal{U}, \forall u \in \mathcal{V}, t \perp u$.

Remark 14. — If $t\{x\backslash v\} \perp u\{x\backslash v\}$, then $t \perp u$ and $\mu x.\{t \mid u\} \in \mathsf{SN}$.

Definition 7. A set $\mathcal{U}$ of Curry-style terms is *simple* if it is non-empty and it contains no Curry-style term of the form $\mu x.p$.

Definition 8. A pair $(\mathcal{U}, \mathcal{V})$ of sets of Curry-style terms is *saturated* if:

- $\mathsf{Var} \subseteq \mathcal{U}$ and $\mathsf{Var} \subseteq \mathcal{V}$
- $\{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{V}, t\{x\backslash v\} \perp u\{x\backslash v\}\} \subseteq \mathcal{U}$ and $\{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{U}, t\{x\backslash v\} \perp u\{x\backslash v\}\} \subseteq \mathcal{V}$.

Definition 9. • Whenever $\mathcal{U}$ is simple, we define the following function $\Phi_{\mathcal{U}}(\mathcal{V}) = \mathcal{U} \cup \mathsf{Var} \cup \{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{V}, t\{x\backslash v\} \perp u\{x\backslash v\}\}$.
• Note that for all simple $\mathcal{U}$, $\Phi_{\mathcal{U}}$ is anti-monotone. Hence, for any simple $\mathcal{U}$ and $\mathcal{V}$, $\Phi_{\mathcal{U}} \circ \Phi_{\mathcal{V}}$ is monotone, so it admits a least fixpoint $\mathcal{U}'$ and we define $\mathsf{FixExt}(\mathcal{U}, \mathcal{V}) = (\mathcal{U}', \Phi_{\mathcal{V}}(\mathcal{U}'))$.

Note that the fixpoint construction is asymmetric: if $\mathsf{FixExt}(\mathcal{U}, \mathcal{V}) = (\mathcal{U}', \mathcal{V}')$, there is *a priori* no reason for $\mathsf{FixExt}(\mathcal{V}, \mathcal{U})$ to be $(\mathcal{V}', \mathcal{U}')$ (the first and second arguments have different roles).

Proposition 15. — *Assume that $\mathcal{U}$ and $\mathcal{V}$ are simple with $\mathcal{U} \perp \mathcal{V}$, and let $(\mathcal{U}', \mathcal{V}') = \mathsf{FixExt}(\mathcal{U}, \mathcal{V})$. We have $\mathcal{U} \subseteq \mathcal{U}'$, $\mathcal{V} \subseteq \mathcal{V}'$, $\mathcal{U}' \perp \mathcal{V}'$ and $(\mathcal{U}', \mathcal{V}')$ is saturated.*

Proof. By definition, we have

$$\mathcal{U}' = \Phi_{\mathcal{U}}(\mathcal{V}') = \mathcal{U} \cup \mathsf{Var} \cup \{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{V}', t\{x\backslash v\} \perp u\{x\backslash v\}\}$$
$$\mathcal{V}' = \Phi_{\mathcal{V}}(\mathcal{U}') = \mathcal{V} \cup \mathsf{Var} \cup \{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{U}', t\{x\backslash v\} \perp u\{x\backslash v\}\}.$$

It is clearly saturated. We now prove that $\mathcal{U}' \perp \mathcal{V}'$.

Since $\mathcal{U} \perp \mathcal{V}$ and $\mathcal{U}$ and $\mathcal{V}$ are non-empty, we have $\mathcal{U} \subseteq \mathsf{SN}$ and $\mathcal{V} \subseteq \mathsf{SN}$. We also have $\mathsf{Var} \subseteq \mathsf{SN}$. Finally, by Remark 14, we conclude $\mathcal{U}' \subseteq \mathsf{SN}$ and $\mathcal{V}' \subseteq \mathsf{SN}$.

Now assume $u \in \mathcal{U}' \subseteq \mathsf{SN}$ and $v \in \mathcal{V}' \subseteq \mathsf{SN}$. If $u \in \mathcal{U}$ and $v \in \mathcal{V}$ then $u \perp v$ because $\mathcal{U} \perp \mathcal{V}$. If not, then at least one of them is a variable or a term of the form $\mu x.p$. In that case we show that for any u' and v' such that $u \longrightarrow^*_{F^c_\omega} u'$ and $v \longrightarrow^*_{F^c_\omega} v'$, we have $u' \perp v'$. Note that $u' \in \mathsf{SN}$ and $v' \in \mathsf{SN}$, and at least one of u' and v' is a variable or a term of the form $\mu x.p'$.

It then suffices to prove that if $\{u' \mid v'\} \longrightarrow_{F^c_\omega} p''$ then $p'' \in \mathsf{SN}$, which we do by lexicographical induction on the length of the longest derivation starting from $u' \in \mathsf{SN}$ and that of the longest derivation starting from $v' \in \mathsf{SN}$.

- If $\{u' \mid v'\} \longrightarrow_{F^c_\omega} \{u'' \mid v'\}$ or $\{u' \mid v'\} \longrightarrow_{F^c_\omega} \{u' \mid v''\}$, the induction hypothesis applies.
- Since at least one of u' and v' is a variable or a term of the form $\mu x.p'$, the only other possible reduction is when $u' = \mu x.p'$ (resp. $v' = \mu x.p'$) and $\{u' \mid v'\} \longrightarrow_{F^c_\omega} p'\{x\backslash v'\}$ (resp. $\{u' \mid v'\} \longrightarrow_{F^c_\omega} p'\{x\backslash u'\}$).
  Since $u \longrightarrow^*_{F^c_\omega} u'$ and $v \longrightarrow^*_{F^c_\omega} v'$, we have $u = \mu x.p$ (resp. $v = \mu x.p$) with $p \longrightarrow^*_{F^c_\omega} p'$, so $p\{x\backslash v\} \longrightarrow^*_{F^c_\omega} p'\{x\backslash v'\}$ (resp. $p\{x\backslash u\} \longrightarrow^*_{F^c_\omega} p'\{x\backslash u'\}$). Since $u \in \mathcal{U}'$ and $v \in \mathcal{V}'$, we know that $p\{x\backslash v\} \in \mathsf{SN}$ (resp. $p\{x\backslash u\} \in \mathsf{SN}$), so $p'\{x\backslash v'\} \in \mathsf{SN}$ (resp. $p'\{x\backslash u'\} \in \mathsf{SN}$). □

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Definition 10. — Now we interpret kinds:

- The interpretation $[\![K]\!]$ of a kind K is defined by induction on K as follows:

$$\begin{array}{lcl} [\![\star]\!] & = & \{(\mathcal{U},\mathcal{V}) \mid \mathcal{U} \perp \mathcal{V} \text{ and } (\mathcal{U},\mathcal{V}) \text{ is saturated}\} \\ [\![K \to K']\!] & = & [\![K']\!]^{[\![K]\!]} \end{array}$$

  where $[\![K']\!]^{[\![K]\!]}$ is simply the set of (total) functions from $[\![K]\!]$ to $[\![K']\!]$.
- Given a pair $p \in [\![\star]\!]$, we write p^+ (resp. p^-) its first (resp. second) component.
- We also define the (involutive) function $\mathsf{swap}_K : [\![K]\!] \to [\![K]\!]$ by induction on K:

$$\begin{array}{ll} \mathsf{swap}_\star(\mathcal{U},\mathcal{V}) & = (\mathcal{V},\mathcal{U}) \\ \mathsf{swap}_{K\to K'}(f) & = \mathsf{swap}_{K'} \circ f. \end{array}$$

- Let $\mathsf{swap} : (\bigcup_K [\![K]\!]) \to (\bigcup_K [\![K]\!])$ be the disjoint union of all the swap_K.

Remark 16. — Given $p \in [\![\star]\!]$, $(\mathsf{swap}_\star(p))^+ = p^-$ and $(\mathsf{swap}_\star(p))^- = p^+$.

Definition 11. — Let $\mathcal{U}$ and $\mathcal{V}$ be sets of Curry-style terms. We set the following definitions:

$$\begin{array}{lcl} \langle \mathcal{U},\mathcal{V}\rangle & = & \{\langle u, v\rangle \mid u \in \mathcal{U}, v \in \mathcal{V}\} \\ \gamma\mathcal{U}\mathcal{V}.\bullet & = & \{\gamma x\, y.p \mid \forall u \in \mathcal{U}\ \forall v \in \mathcal{V}\ p\{x, y\backslash u, v\} \in \mathsf{SN}\} \\ \Lambda\_.\mathcal{U} & = & \{\Lambda\_.u \mid u \in \mathcal{U}\} \\ \langle \_, \mathcal{U}\rangle & = & \{\langle \_, u\rangle \mid u \in \mathcal{U}\}. \end{array}$$

Remark 17. 1. The sets $\langle \mathcal{U},\mathcal{V}\rangle$, $\gamma\mathcal{U}\mathcal{V}.\bullet$, $\Lambda\_.\mathcal{U}$ and $\langle \_, \mathcal{U}\rangle$ are always simple.
2. If $\mathcal{U} \subseteq \mathsf{SN}$ and $\mathcal{V} \subseteq \mathsf{SN}$ then $\langle \mathcal{U},\mathcal{V}\rangle \perp \gamma\mathcal{U}\mathcal{V}.\bullet$.
3. If $\mathcal{U} \perp \mathcal{V}$ then $\Lambda\_.\mathcal{U} \perp \langle \_, \mathcal{V}\rangle$.

Definition 12. — We say that a mapping $\rho : \mathsf{Var}^T \to \bigcup_K [\![K]\!]$ is *compatible* with Σ if $\forall(\alpha : K) \in \Sigma$, $\rho(\alpha) \in [\![K]\!]$.

Definition 13. — For each A such that $\Sigma \vdash A : K$ for some K, and for each ρ compatible with Σ, we define $[\![A]\!]_\rho \in [\![K]\!]$ as follows:

$$\begin{array}{lcl} [\![\alpha]\!]_\rho & = & \rho(\alpha) \\ [\![\alpha^\perp]\!]_\rho & = & \mathsf{swap}([\![\alpha]\!]_\rho) \\ [\![A \wedge B]\!]_\rho & = & \mathsf{FixExt}(\langle [\![\mathsf{A}]\!]^+_\rho, [\![\mathsf{B}]\!]^+_\rho\rangle, \gamma[\![\mathsf{A}]\!]^+_\rho [\![\mathsf{B}]\!]^+_\rho.\bullet) \\ [\![A \vee B]\!]_\rho & = & \mathsf{swap}(\mathsf{FixExt}(\langle [\![\mathsf{A}]\!]^-_\rho, [\![\mathsf{B}]\!]^-_\rho\rangle, \gamma[\![\mathsf{A}]\!]^-_\rho [\![\mathsf{B}]\!]^-_\rho.\bullet)) \\ [\![\forall\alpha : K'.\, A]\!]_\rho & = & \mathsf{FixExt}(\Lambda\_.\bigcap_{\mathsf{h}\in[\![\mathsf{K}']\!]} [\![\mathsf{A}]\!]^+_{\rho,\alpha\mapsto\mathsf{h}}, \langle \_, \bigcup_{\mathsf{h}\in[\![\mathsf{K}']\!]} [\![\mathsf{A}]\!]^-_{\rho,\alpha\mapsto\mathsf{h}}\rangle) \\ [\![\exists\alpha : K'.\, A]\!]_\rho & = & \mathsf{swap}(\mathsf{FixExt}(\Lambda\_.\bigcap_{\mathsf{h}\in[\![\mathsf{K}']\!]} [\![\mathsf{A}]\!]^-_{\rho,\alpha\mapsto\mathsf{h}}, \langle \_, \bigcup_{\mathsf{h}\in[\![\mathsf{K}']\!]} [\![\mathsf{A}]\!]^+_{\rho,\alpha\mapsto\mathsf{h}}\rangle)) \\ [\![\lambda\alpha : K'.\, A]\!]_\rho & = & h \in [\![K']\!] \mapsto [\![A]\!]_{\rho,\alpha\mapsto h} \\ [\![A\ B]\!]_\rho & = & ([\![A]\!]_\rho)([\![B]\!]_\rho). \end{array}$$

The soundness of the definition inductively relies on the facts that ρ keeps being compatible with Σ and $[\![A]\!]_\rho \in [\![K]\!]$ (using Remark 17 and Proposition 15). In particular if $\Sigma \vdash A : \star$, then $[\![A]\!]_\rho$ is orthogonal and saturated (with $[\![A]\!]^+_\rho \subseteq \mathsf{SN}$ and $[\![A]\!]^-_\rho \subseteq \mathsf{SN}$).

Remark 18. 1. $[\![A^\perp]\!]_\rho = \mathsf{swap}([\![A]\!]_\rho)$.
2. $[\![A\{\alpha\backslash B\}]\!]_\rho = [\![A]\!]_{\rho,\alpha\mapsto[\![B]\!]_\rho}$
3. If $A \longrightarrow_\beta B$ then $[\![A]\!]_\rho = [\![B]\!]_\rho$.

Note that the swapping operation, in the interpretation of disjunction and existential quantification, ensures that the *same* fixpoint extensions are used in the interpretation of A and in that of $A^\perp$. This is necessary for establishing the above remark, given that, *a priori*, the fixpoint extension is asymmetric (i.e. $\mathsf{swap}_\star \circ \mathsf{FixExt} \neq \mathsf{FixExt} \circ \mathsf{swap}_\star$). The choice of using swapping for disjunction (rather than conjunction) and existential (rather than universal) quantification is arbitrary, and, correspondingly, we could have defined $\mathsf{FixExt}(\mathcal{U},\mathcal{V})$ with the fixpoint of $\Phi_\mathcal{V} \circ \Phi_\mathcal{U}$ rather than $\Phi_\mathcal{U} \circ \Phi_\mathcal{V}$. Alternatively we could also have used greatest fixpoints.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

Proposition 19. — *If* $x_1 : A_1, \ldots, x_n : A_n \vdash_\Sigma t : A$ *then for all* ρ *compatible with* Σ*, and for all* $t_1 \in [\![A_1]\!]^+_\rho, \ldots, t_n \in [\![A_n]\!]^+_\rho$ *we have:*

$$\|t\|\{x_1, \ldots, x_n \backslash t_1, \ldots, t_n\} \in [\![A]\!]^+_\rho.$$

Proof. By induction on the typing tree. □

Theorem 20. — *If* $x_1 : A_1, \ldots, x_n : A_n \vdash_\Sigma t : A$ *then* $t \in \mathcal{SN}^{F^c_\omega}$.

Proof. We first prove that we can find a ρ compatible with Σ (for $\alpha : \star$, take $\rho(\alpha) = \mathsf{FixExt}(\mathsf{Var}, \mathsf{Var})$). Then we can apply Proposition 19 and conclude by Lemma 13. □

4. Orthogonality and saturation

As mentioned in the introduction of Section 3, the similarity between the proof of strong normalisation of the upper layer and that of the lower layer is striking.

However, while in the upper layer the saturation of the interpretation of kinds is obtained by a bi-orthogonal completion, it is important to understand why, for the lower layer, we used another notion of completion using fixpoints instead.

The reason is that in general, if the pair $(\mathcal{U}, \mathcal{V})$ is simple and orthogonal, the extension $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ might not be saturated in the sense of Definition 8 (while in the upper layer such a completion by bi-orthogonality ensures the corresponding notion of saturation). This was a conjecture set in [15], which we prove in this section by providing counter-examples.

Technically, the presence of the μ–μ critical pair makes the proof of Theorem 9.3 impossible to adapt to the non-confluent case of the lower layer. This lack of saturation is the motivation for the fixpoint construction in the interpretation of types, instead of the bi-orthogonal construction.

Note that [11] already notices that 'the technique using the usual candidates of reducibility does not work' for the non-confluent reductions of classical logic (that they express in the $\lambda\mu$-calculus [18]). However, their counter-examples translate in our setting to the fact that even if t and $p\{x\backslash t\}$ are in SN, $\{\mu x.p \mid t\}$ need not be in SN. This is quite direct, but the method of completion by bi-orthogonality is more subtle: Indeed, we claim here that a bi-orthogonal extension $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ (with $\mathcal{V}^{\perp\perp} = \mathcal{U}^\perp$ and $\mathcal{U}^{\perp\perp} = \mathcal{V}^\perp$) need not be saturated. In other words, there exist $t \in \mathcal{V}^{\perp\perp}$ and $p\{x\backslash t\} \in \mathsf{SN}$, such that $\mu x.p \notin \mathcal{U}^{\perp\perp}$ (or the symmetric situation, swapping $\mathcal{U}$ and $\mathcal{V}$). Indeed, we do obtain this from $\{\mu x.p \mid t\} \notin \mathsf{SN}$, but the counter-examples of [11] only provide this with $t \in \mathsf{SN}$ instead of $t \in \mathcal{V}^{\perp\perp} \subseteq \mathsf{SN}$.

4.1. A counter-example

Remark 21. — We have the following equivalences for all programs p, q and for all terms t:

1. $\{\mu x.p \mid \mu y.q\} \in \mathsf{SN}$ iff $p\{x\backslash\mu y.q\} \in \mathsf{SN}$ and $q\{y\backslash\mu x.p\} \in \mathsf{SN}$.
2. If the term t is not a μ-abstraction, then
   $\{\mu x.p \mid t\} \in \mathsf{SN}$ iff $t \in \mathsf{SN}$ and $p\{x\backslash t\} \in \mathsf{SN}$.

We write $p + q$ for the non-deterministic composition of programs $\{\mu\_.p \mid \mu\_.q\}$. (where $\_$ denotes any fresh variable), which reduces to both p and q. We have the equivalence:

$$(p + q) \in \mathsf{SN} \quad \text{iff} \quad p \in \mathsf{SN} \quad \text{and} \quad q \in \mathsf{SN}.$$

Let $\delta = \mu x.\{x \mid x\}$. The counter-example is the following:

Proposition 22 (*Counter-Example to Saturation*).
— *The pair* $(\{\delta\}^\perp, \{\delta\}^{\perp\perp})$ *is not saturated.*

To prove this proposition, let us consider the program

$$p = \{x \mid a\} + \{x \mid b\},$$

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

where a and b are two normal terms such that

$$\{a, b\} \perp \delta \qquad \text{and} \qquad a \not\perp b.$$

Obvious choices for a and b are $\langle \delta, \delta \rangle$ and $\Upsilon x_1 x_2.\{x_1 \mid x_2\}$, respectively.

Lemma 23. — *For all $t \in \{\delta\}^{\perp\perp}$, we have $p\{x\backslash t\} \in \mathsf{SN}$.*

Proof. Let $t \in \{\delta\}^{\perp\perp}$. Since $a, b \in \{\delta\}^{\perp}$, we have $\{t \mid a\} \in \mathsf{SN}$ and $\{t \mid b\} \in \mathsf{SN}$, hence $p\{x\backslash t\} = \{t \mid a\} + \{t \mid b\} \in \mathsf{SN}$ from Proposition 21. □

Lemma 24. — $\mu x.p \notin \{\delta\}^{\perp}$.

Proof. Reduction of $\{\mu x.p \mid \delta\}$ yields the following sequence:

$$\begin{array}{lll} \{\mu x.p \mid \delta\} & \longrightarrow_{F_\omega^c} & \{\mu x.p \mid \mu x.p\} \\ & \longrightarrow_{F_\omega^c} & \{\mu x.p \mid a\} + \{\mu x.p \mid b\} \\ & \longrightarrow_{F_\omega^c} & \{\mu x.p \mid a\} \\ & \longrightarrow_{F_\omega^c} & \{a \mid a\} + \{a \mid b\} \\ & \longrightarrow_{F_\omega^c} & \{a \mid b\} \quad \notin \quad \mathsf{SN}, \end{array}$$

hence $\mu x.p \notin \{\delta\}^{\perp}$. □

Lemmas 23 and 24 complete the proof of Proposition 22.

4.2. *Perfect normalisation and a refined counter-example*

The counter-example presented in Section 4.1 relies on two terms a and b that are orthogonal to δ, that is, such that

$$\{a \mid \delta\} \in \mathsf{SN} \qquad \text{and} \qquad \{b \mid \delta\} \in \mathsf{SN}.$$

It is interesting to notice that for the choice of a and b we gave above, the strong normalisation of both programs $\{a \mid \delta\}$ and $\{b \mid \delta\}$ relies on the fact that all reduction sequences eventually block on an incestuous program:

$$\{a \mid \delta\} = \{\langle \delta, \delta \rangle \mid \delta\} \longrightarrow_{F_\omega^c} \{\langle \delta, \delta \rangle \mid \langle \delta, \delta \rangle\} \qquad \text{and}$$
$$\{b \mid \delta\} = \{\Upsilon x_1 x_2.\{x_1 \mid x_2\} \mid \delta\} \longrightarrow_{F_\omega^c} \{\Upsilon x_1 x_2.\{x_1 \mid x_2\} \mid \Upsilon x_1 x_2.\{x_1 \mid x_2\}\}.$$

Of course, the computations above should be considered as ill-typed in any reasonable typing system, and thus should be rejected.

On the other hand, the orthogonality relation $t \perp u$ is intended to express some kind of correctness about the execution of the program $\{t \mid u\}$. Thus if we consider that the strong normalisation of $\{a \mid \delta\}$ and $\{b \mid \delta\}$ is purely artificial, one should restrict the definition of orthogonality in such a way that the pairs (a, δ) and (b, δ) are rejected. This naturally leads to the following definition:

Definition 14 (*Perfectly Normalising Program*). — A program p (resp. a term t) is said to be *perfectly normalising* if it is strongly normalising, and if for all p' such that $p \longrightarrow_{F_\omega^c}^* p'$ (resp. all t' such that $t \longrightarrow_{F_\omega^c}^* t'$), the program p' (the term t') contains no incestuous program as a subterm.

The set of all perfectly normalising programs and terms – which is a subset of the set SN of all strongly normalising programs and terms – is written PN. Perfect normalisation enjoys similar properties as strong normalisation:

Remark 25. — We have the following equivalences for all programs p, q and for all terms t:

1. $\{\mu x.p \mid \mu y.q\} \in \mathsf{PN}$ iff $p\{x\backslash \mu y.q\} \in \mathsf{PN}$ and $q\{y\backslash \mu x.p\} \in \mathsf{PN}$.
2. If the term t is not a μ-abstraction, then
   $\{\mu x.p \mid t\} \in \mathsf{PN}$ iff $t \in \mathsf{PN}$ and $p\{x\backslash t\} \in \mathsf{PN}$.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

The notion of perfect normalisation induces a new orthogonality relation —still written $t \perp u$— on the set PN of perfectly normalising terms, setting:

$$t \perp u = \{t \mid u\} \in \mathsf{PN}.$$

In this setting, the counter-example of Section 4.1 does not work anymore, since $a, b \notin \{\delta\}^{\perp}$ (using the new definition of the operator $\mathcal{U} \mapsto \mathcal{U}^{\perp}$).

Thus, we can still wonder whether pairs of sets of terms of the form $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ (according to the new definition of orthogonality) are always saturated or not.

Again, the answer is negative, but the counter-example is more subtle. We replace the symmetric self-application $\delta = \mu x.\{x \mid x\}$ by a notion of self-application coming from the λ-calculus

$$\delta = \mu x.\{x \mid \langle x, z\rangle\},$$

where z denotes a fixed free variable.

Proposition 26. — *The pair $(\{\delta\}^{\perp}, \{\delta\}^{\perp\perp})$ is not saturated. (Where $\perp$ refers to perfectly normalising orthogonality.)*

Again, the idea is to consider two terms a and b such that $\{a \mid \delta\} \in \mathsf{PN}$, $\{b \mid \delta\} \in \mathsf{PN}$ (intuitively: the λ-terms aa and bb strongly normalise), but such that $\{b \mid \langle a, z\rangle\} \notin \mathsf{PN}$ (intuitively: the λ-term ba diverges). For that, consider the following terms

$$\begin{array}{ll} \Delta = \Upsilon x\, y.\{x \mid \langle x, y\rangle\} & (\approx \lambda x.\, xx) \\ a = \Upsilon\_ y.\{\Delta \mid y\} & (\approx \mathbf{K}\Delta) \\ b = \Upsilon x\, y.\{x \mid \langle z, \langle \Delta, y\rangle\rangle\} & (\approx \lambda x.\, xz\Delta) \end{array}$$

(where $\mathbf{K} = \lambda x y.\, x$) and set again

$$p = \{a \mid x\} + \{b \mid x\}.$$

Lemma 27. — *For all $t \in \{\delta\}^{\perp\perp}$, we have $p\{x\backslash t\} \in \mathsf{PN}$.*

Proof. In order to check that $\{a \mid \delta\} \in \mathsf{PN}$ and $\{b \mid \delta\} \in \mathsf{PN}$, we now have to check that these programs do not reduce to programs containing incestuous pairs. Indeed, the only reductions of these programs are:

$$\begin{array}{l} \{a \mid \delta\} \longrightarrow_{F_\omega^c} \{a \mid \langle a, z\rangle\} \longrightarrow_{F_\omega^c} \{\Delta \mid z\} \\ \{b \mid \delta\} \longrightarrow_{F_\omega^c} \{b \mid \langle b, z\rangle\} \longrightarrow_{F_\omega^c} \{b \mid \langle z, \langle \Delta, z\rangle\rangle\} \longrightarrow_{F_\omega^c} \{z \mid \langle z, \langle \Delta, \langle \Delta, z\rangle\rangle\rangle\}. \end{array}$$

Hence $a, b \in \{\delta\}^{\perp}$. Assume that $t \in \{\delta\}^{\perp\perp}$. We thus have $\{a \mid t\} \in \mathsf{PN}$ and $\{b \mid t\} \in \mathsf{PN}$, hence $\{a \mid t\} + \{b \mid t\} = p\{x\backslash t\} \in \mathsf{PN}$ from Proposition 25. □

Lemma 28. — $\mu x.p \notin \{\delta\}^{\perp}$.

Proof. Reduction of $\{\mu x.p \mid \delta\}$ yields the following sequence:

$$\begin{array}{ll} \{\mu x.p \mid \delta\} & \longrightarrow_{F_\omega^c} \{\mu x.p \mid \langle \mu x.p, z\rangle\} \\ & \longrightarrow_{F_\omega^c} \{a \mid \langle \mu x.p, z\rangle\} + \{b \mid \langle \mu x.p, z\rangle\} \\ & \longrightarrow_{F_\omega^c} \{b \mid \langle \mu x.p, z\rangle\} \\ & \longrightarrow_{F_\omega^c} \{\mu x.p \mid \langle z, \langle \Delta, z\rangle\rangle\} \\ & \longrightarrow_{F_\omega^c} \{a \mid \langle z, \langle \Delta, z\rangle\rangle\} + \{b \mid \langle z, \langle \Delta, z\rangle\rangle\} \\ & \longrightarrow_{F_\omega^c} \{a \mid \langle z, \langle \Delta, z\rangle\rangle\} \\ & \longrightarrow_{F_\omega^c} \{\Delta \mid \langle \Delta, z\rangle\} \\ & \longrightarrow_{F_\omega^c} \{\Delta \mid \langle \Delta, z\rangle\} \notin \mathsf{PN} \end{array}$$

hence $\mu x.p \notin \{\delta\}^{\perp}$. □

Lemmas 27 and 28 complete the proof of Proposition 26.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

5. Logical properties

5.1. Consistency

The consistency of F^{c}_{ω} follows from Theorem 20 using a simple combinatorial argument. Let us first notice that all (untyped) programs that are in normal form are either incestuous pairs or programs of the following forms:

VARIABLE–VARIABLE	$\{x \mid y\}$
VARIABLE–LAMBDA	$\{x \mid \gamma x^A y^B . p\}$
VARIABLE–PAIR	$\{x \mid \langle t, u\rangle\}$
VARIABLE–∀LAMBDA	$\{x \mid \Lambda\alpha : K . t\}$
VARIABLE–∃WITNESS	$\{x \mid \langle A, t\rangle\}$.

Lemma 29. *There is no closed typed program in normal form.*

Proof. As mentioned in Proposition 8, incestuous pairs cannot be typed, and all programs of one of the above four forms have a free variable, namely x. □

Hence we get the logical consistency of system F^{c}_{ω}.

Theorem 30 (*Consistency*). *There is no closed typed program in* F^{c}_{ω}.

Proof. It suffices to combine Lemma 29 with Theorem 20 and Proposition 7. □

5.2. Translating F_ω + DNE into F^{c}_{ω}

The definition of implication $A \Rightarrow B$ as $(A^{\perp}) \vee B$ naturally suggests a translation from system F_ω to system F^{c}_{ω}. We annotate sequents in F_ω using $\vdash^{F_\omega}$.

The translation proceeds as follows: each kind of F_ω is translated as itself, and each type constructor A of F_ω is translated as a type constructor A^* of F^{c}_{ω} by the equations

$$\begin{aligned}
\alpha^* &= \alpha \\
(\forall\alpha : K . A)^* &= \forall\alpha : K . A^* \\
(A \Rightarrow B)^* &= A^{*\perp} \vee B^* \\
(\lambda\alpha : K . B)^* &= \lambda\alpha : K . B^* \\
(B\ A)^* &= B^*\ A^*.
\end{aligned}$$

We then easily check that

Proposition 31. — *If* $\Sigma \vdash^{F_\omega} A : K$, *then* $\Sigma \vdash A^* : K$.

Proposition 32. — *If* $A \longrightarrow_\beta B$, *then* $A^* \longrightarrow_\beta B^*$.

We now translate proof-terms, adapting Prawitz's translation of natural deduction into sequent calculus, this time using Curry-style terms and programs, because without a typing derivation for the terms of F_ω we lack some type annotations to place in the encoding.

Definition 15 (*Encoding of Terms*). The encoding u^* of a term u of F_ω is defined by induction on u as described in Fig. 6. It relies on an auxiliary encoding that maps u to a program u^*_t and that is parameterised by a term t of F^{c}_{ω}.

Remark 33. — Let t, t' be two terms of F^{c}_{ω}, and u, u' two terms of F_ω.

1. If $t \longrightarrow_{F^{\mathsf{c}}_{\omega}} t'$ then $u^*_t \longrightarrow_{F^{\mathsf{c}}_{\omega}} u^*_{t'}$.
2. $\{u^* \mid t\} \longrightarrow^*_{F^{\mathsf{c}}_{\omega}} u^*_t$
3. $u^*_t\{x\backslash u'^*\} \longrightarrow^*_{F^{\mathsf{c}}_{\omega}} u\{x\backslash u'\}^*_{t\{x\backslash u'^*\}}$ and
   $u^*\{x\backslash u'^*\} \longrightarrow^*_{F^{\mathsf{c}}_{\omega}} u\{x\backslash u'\}^*$.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

$$
\begin{array}{lcll}
x^* & = & x & \\
\lambda x^A.u^* & = & \Upsilon x\, y.u^*_y & \\
\Lambda\alpha : K\ .\ u^* & = & \Lambda\_.u^* & \\
u^* & = & \mu y.u^*_y & \text{otherwise} \\
\hline
(u\ u')^*_t & = & u^*_{\langle u'^*, t\rangle} & \\
(u\ A)^*_t & = & u^*_{\langle \_, t\rangle} & \\
v^*_t & = & \{v^* \mid t\} & \text{otherwise}
\end{array}
$$

Fig. 6. Encoding of terms.

The encoding of terms allows the simulation of reductions:

Proposition 34 (*Simulation of β for Terms*).
If $u \longrightarrow_{F_\omega} u'$, then $u^*_t \longrightarrow^+_{F^{\mathsf{c}}_\omega} u'^*_t$ and $u^* \longrightarrow^+_{F^{\mathsf{c}}_\omega} u'^*$.

Proof. By simultaneous induction on the derivation of the reduction step, using Remark 33. □

The translation preserves typing:

Proposition 35 (*Preservation of Typing for Terms*). 1. *If $\Gamma \vdash^{F_\omega}_{\Sigma} u : A$, then there exists a term t of system F^{c}_ω (with type annotations) such that $\|t\| = u^*$ and $\Gamma^* \vdash_\Sigma t : A$.*
2. *If $\Gamma \vdash^{F_\omega}_{\Sigma} u : A$ and $\Gamma^*, \Delta \vdash_\Sigma t : A^{*\perp}$, then there exists a program p of system F^{c}_ω (with type annotations) such that $\|p\| = u^*_{\|t\|}$ and $\Gamma^*, \Delta \vdash_\Sigma p\ \diamond$.*

Proof. By induction on derivations, using Theorem 32 for the conversion rule. □

Since F^{c}_ω is classical, we have a proof of the axiom of double negation elimination:

Let $\bot = \forall\alpha:\star.\alpha$ (in F_ω and F^{c}_ω) and $\top = \exists\alpha:\star.\alpha$ (in F^{c}_ω), and let DNE be the proposition $\forall\alpha:\star.((\alpha \Rightarrow \bot) \Rightarrow \bot) \Rightarrow \alpha$ expressed in system F_ω. We have $\mathrm{DNE}^* = \forall\alpha:\star.((\alpha^\perp \vee \bot) \wedge \top) \vee \alpha$. Let $\mathsf{C} = \Lambda\alpha:\star.\,\Upsilon x^B y^{\alpha^\perp}.\{x \mid \langle \Upsilon x'^\alpha y'^\top.\{x' \mid y\}, \langle \alpha^\perp, y\rangle\rangle\}$, where $B = (\alpha \wedge \top) \vee \bot$.

We have

$$\vdash \mathsf{C} : \mathrm{DNE}^*.$$

Hence, provable propositions of system $F_\omega + \mathrm{DNE}$ become provable propositions of system F^{c}_ω:

Theorem 36 (F^{c}_ω *Captures* $F_\omega + \mathrm{DNE}$). *For all derivable judgements of the form*

$$z : \mathrm{DNE},\ \Gamma \vdash^{F_\omega}_{\Sigma} u : A$$

there exists a term t of system F^{c}_ω (with type annotations) such that $\|t\| = u^*$ and we have

$$\Gamma^* \vdash_\Sigma t\{z \backslash \mathsf{C}\} : A^*.$$

Through the translation $A \mapsto A^*$, system F^{c}_ω appears as an extension of system $F_\omega + \mathrm{DNE}$, and hence the consistency of F^{c}_ω, proved in Section 5.1, implies that of $F_\omega + \mathrm{DNE}$.

We then set the following conjecture:

Conjecture 37 (F^{c}_ω *is a Conservative Extension of* $F_\omega + \mathrm{DNE}$).
There exists a mapping $\mathcal{B}$ of the upper layer of F^{c}_ω into that of F_ω such that:

1. *If $\Sigma \vdash^{F_\omega} A : \star$, then there exist two terms u and u' such that $\vdash^{F_\omega}_{\Sigma} u : A \to \mathcal{B}(A^*)$ and $\vdash^{F_\omega}_{\Sigma} u' : \mathcal{B}(A^*) \to A$.*
2. *If $\Gamma \vdash_\Sigma t : A$ then there exists a term u of F_ω such that $\mathcal{B}(\Gamma), z : \mathrm{DNE} \vdash_\Sigma u : \mathcal{B}(A)$.*

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

6. Conclusion

In this paper we have introduced a classical version of system F_ω, called F^{c}_ω. Its upper layer is intuitionistic, its lower layer is classical, and both are strongly normalising.

We have adapted Tait and Girard's reducibility methods for the two strong normalisation results, using orthogonality and, for the lower layer, Barbanera and Berardi's symmetric candidates.

F^{c}_ω thus provides an opportunity to compare the two variants of the reducibility method, which we do in Section 4, proving the conjecture set in [15] that orthogonality does not capture the fixpoint completion of the symmetric candidates. It is worth noting that the counter-examples are not specific to F^{c}_ω at all. First, they hold in propositional logic (they do not involve polymorphism or type constructors), and second they could easily be given for other symmetric calculi for classical logic such as the symmetric λ-calculus [4], the $\overline{\lambda}\mu\widetilde{\mu}$-calculus [8] or the dual calculus [23], as long as their untyped versions feature some infinite computations related to the λ-term $\Delta\Delta$.

This point being made, it is clear that alternative proofs could have been given instead. For the upper layer we could simply have simulated the reduction in the simply typed λ-calculus, forgetting all the information about duality (A and $A^\perp$ would be mapped to the same term) which plays no computational role in this layer.[5]

However, such an encoding, while preserving the notion of computation, loses all information about duality. This has two consequences:

- It cannot be used to establish a reflection between the upper layer of F^{c}_ω and the simply typed λ-calculus (or the upper layer of F_ω).
- Since it loses all the logical meaning of type constructors, it cannot be used for a type-preserving encoding of F^{c}_ω into e.g. F_ω+DNE, which we need to prove the conservativity conjecture (Conjecture 37 of Section 5.2).

Ongoing work is about refining this forgetful mapping by encoding in λ-terms the information about duality, i.e. some notion of "polarity", in a way that is useful for the above two points.

For the lower layer we could try to adapt to F_ω simpler proofs of strong normalisation of symmetric and non-confluent calculi for classical logic, such as those of [10] or [12] which do not involve the fixpoint construction. We do not know whether these proofs break, for a typing system as strong as that of F^{c}_ω. While we have seen that the fixpoint completion is not captured by orthogonality, it would be interesting to see whether these simpler proofs are captured by it (although they are not expressed in the framework of reducibility to which orthogonality pertains).

References

[1] H.P. Barendregt, The Lambda-calculus, its syntax and semantics, in: Studies in Logic and the Foundation of Mathematics, second ed., Elsevier, 1984.
[2] H.P. Barendregt, Introduction to generalized type systems, J. Funct. Programming 1 (2) (1991) 125–154.
[3] H.P. Barendregt, Lambda calculi with types, in: S. Abramsky, D.M. Gabby, T.S.E. Maibaum (Eds.), Hand. Log. Comput. Sci., vol. 2, Oxford University Press, 1992, pp. 117–309 (Chapter 2).
[4] F. Barbanera, S. Berardi, A symmetric lambda-calculus for classical program extraction, Inform. Comput. 125 (2) (1996) 103–117.
[5] H. Barendregt, H. Geuvers, Proof-assistants using dependent type systems, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Elsevier and MIT Press, 2001, pp. 1149–1238.
[6] G. Barthe, J. Hatcliff, M.H. Sørensen, A notion of classical pure type system, in: S. Brookes, M. Main, A. Melton, M. Mislove (Eds.), Proc. of the 13th Annual Conf. on Math. Foundations of Programming Semantics, MFPS'97, in: ENTCS, vol. 6, Elsevier, 1997, pp. 4–59.
[7] M. Coppo, M. Dezani-Ciancaglini, A new type assignment for lambda-terms, Arch. f. Math. Logic u. Grundlagenforschung 19 (1978) 139–156.
[8] P.-L. Curien, H. Herbelin, The duality of computation, in: Proc. of the 5th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'00, ACM Press, 2000, pp. 233–243.
[9] D.J. Dougherty, S. Ghilezan, P. Lescanne, S. Likavec, Strong normalization of the dual classical sequent calculus, in: G. Sutcliffe, A. Voronkov (Eds.), Proc. of the 12th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'05, in: LNCS, vol. 3835, Springer-Verlag, December 2005, pp. 169–183.
[10] R. David, K. Nour, Arithmetical proofs of strong normalization results for the symmetric $\lambda\mu$, in: P. Urzyczyn (Ed.), Proc. of the 9th Int. Conf. on Typed Lambda Calculus and Applications, TLCA'05, in: LNCS, vol. 3461, Springer-Verlag, April 2005, pp. 162–178.

[5] For instance, α and $\alpha^\perp$ would be mapped to the same term, $A \wedge B$ and $A \vee B$ would both be mapped to $x_{\wedge\vee}\ A\ B$ and $\forall\alpha : K\ .\ B$ and $\exists\alpha : K\ .\ A$ would both be mapped to $x_{\forall\exists}\ \lambda\alpha.A$ for two particular variables $x_{\wedge\vee}$ and $x_{\forall\exists}$ that are never bound because they represent the logical connectives.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005

[11] R. David, K. Nour, Why the usual candidates of reducibility do not work for the symmetric $\lambda\mu$-calculus, in: P. Lescanne, R. David, M. Zaionc (Eds.), Post-proc. of the 2nd Work. on Computational Logic and Applications, CLA'04, in: ENTCS, vol. 140, Elsevier, 2005, pp. 101–111.
[12] D. Dougherty, Personal communication, August 2006.
[13] J.-Y. Girard, Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse d'état, Université Paris 7, 1972.
[14] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1) (1987) 1–101.
[15] S. Lengrand, A. Miquel, A classical version of F_ω, in: S. van Bakel, S. Berardi (Eds.), 1st Work. on Classical logic and Computation, July 2006.
[16] P. Martin-Löf, Intuitionistic type theory, in: Number 1 in Studies in Proof Theory, Lecture Notes, Bibliopolis, 1984.
[17] P.-A. Melliès, J. Vouillon, Recursive polymorphic types and parametricity in an operational framework, in: P. Panangaden (Ed.), 20th Annual IEEE Symp. on Logic in Computer Science, IEEE Computer Society Press, June 2005, pp. 82–91.
[18] M. Parigot, $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction, in: A. Voronkov (Ed.), Proc. of the Int. Conf. on Logic Programming and Automated Reasoning, LPAR'92, in: LNCS, vol. 624, Springer-Verlag, July 1992, pp. 190–201.
[19] E. Polonovski, Strong normalization of $\lambda\mu\tilde{\mu}$-calculus with explicit substitutions, in: I. Walukiewicz (Ed.), Proc. of the 7th Int. Conf. on Foundations of Software Science and Computation Structures, FOSSACS'04, in: LNCS, vol. 2987, Springer-Verlag, March 2004, pp. 423–437.
[20] P. Selinger, Control categories and duality: on the categorical semantics of the $\lambda\mu$-calculus, Math. Struct. Comput. Sci. 11 (2001) 207–260.
[21] C.A. Stewart, On the formulae-as-types correspondence for classical logic, Ph.D. Thesis, University of Oxford, 2000.
[22] C. Urban, Classical Logic and Computation, Ph.D. Thesis, University of Cambridge, 2000.
[23] P. Wadler, Call-by-value is dual to call-by-name, in: Proc. of the 8th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'03, vol. 38, ACM Press, September 2003, pp. 189–201.

Please cite this article in press as: S. Lengrand, A. Miquel, Classical F_ω, orthogonality and symmetric candidates, Annals of Pure and Applied Logic (2008), doi:10.1016/j.apal.2008.01.005