Abstract. We consider a simple model of higher order, functional computations over the booleans. Then, we enrich the model in order to encompass non-termination and unrecoverable errors, taken separately or jointly. We show that the models so defined form a lattice when ordered by the \textit{extensional collapse situation} relation, introduced in order to compare models with respect to the amount of “intensional information” that they provide on computation. The proofs are carried out by exhibiting suitable applied $\lambda$-calculi, and by exploiting the fundamental lemma of logical relations.

1 Introduction

Properties of programs are often considered as split in two categories: the \textit{extensional} properties and the \textit{intensional} ones. Typical intensional properties are those concerning complexity issues, while typical extensional ones are those concerning input-output, observable, behaviours. This dichotomy originated from recursion theory: a predicate $P(M)$ on Turing Machines is extensional if, whenever it holds for a T.M. $M$, it holds for all the T.M. computing the same function as $M$. Otherwise, $P$ is intensional. So, for instance, whether the computation on 0 takes more or less than 100 steps is an intensional property, while undefinedness on 0 is an extensional one.

It is the intended meaning of a T.M., namely the underlying partial recursive function, which determines the frontier between what is (to be considered as) extensional and what is (to be considered as) intensional.

Therefore, a tentative definition of the notion of extensional property could be the following: a property of programs is extensional if the fact whether or not a program $P$ enjoys it does affect the denotation of $P$; it is intensional otherwise.

Stated this way, being extensional or intensional is a matter of the intended meaning of programs, and, as such, it is a relative notion\footnote{The very same happens for the dichotomy “function vs algorithm”, which could be misleadingly considered as an absolute one. For instance, the well-known parallel-or “function” is an algorithm relatively to the model $S$ of \textit{total objects}, and a function relatively to the model $C$ of \textit{partial objects}, defined below.}.

The following examples, written in a call by name simply typed $\lambda$-calculus with constants, illustrate this claim:
Example 1. The terms:

(1) $\lambda x:\text{bool}. \text{true}$
(2) $\lambda x:\text{bool}. \text{if } x \text{ then true else true}$

get the same interpretation in a model where the type of booleans is interpreted by the set \{\texttt{t}, \texttt{f}\}, and higher types by the corresponding function space in \textit{Set}; they get different interpretations in a model that accommodates non-terminating computations, denoted by a suitable element $\bot$.

Otherwise stated, strictness is an intensional property relatively to the very simple, set-theoretical model, an extensional one relatively to a more sophisticated one.

The linguistic feature that forces strictness to be taken into account extensionally is anything allowing for non-termination (e.g. fixpoint operators).

Example 2. Similarly, the terms:

(3) $\lambda x:\text{bool}. \Omega$
(4) $\lambda x:\text{bool}. \text{if } x \text{ then } \Omega \text{ else } \Omega$

where $\Omega$ is any diverging term, get the same interpretation in the model of non-termination introduced above; they get different interpretations in a model that accommodates also errors, denoted by suitable element $\top$.

More examples could be provided, concerning for instance the property of linearity, or the evaluation-order dependency, but they go beyond the semantic analysis that we propose in the present work.

Hence, the frontier between extensional and intensional properties is determined essentially by the model we refer to. Nevertheless, certain features of programs have to be taken into account extensionally by all models, namely those features that affect the operational behaviour of terms (for instance, strictness in a language allowing for non-termination).

By the way, a fully abstract model is one that keeps implicit (i.e. intensional) all the properties that can be kept implicit.

It appears that the models of simply typed $\lambda$-calculi may be classified with respect to the amount of information on programs that they provide explicitly, i.e. extensionally.

In this paper, we propose a way of performing this classification, and show how it works on some very basic models of higher order computation over the booleans.

The basic tool used for the classification is the notion of extensional collapse situation. An extensional collapse situation is given by two models\(^2\), and a binary logical relation between them which is a partial surjective function, at all types. In this case, the small model, i.e. the target of the surjective function, can be considered as what is left of the big one when some kind of computational behaviour (in this paper, we focus on non-termination and errors) is forbidden.

Elements of the big model may be mapped onto the same element of the small one by the surjection: this is the case, for instance, of the interpretations

\(^2\) The models considered in this paper are families of sets indexed by simple types, usually called type frames.
Fig. 1. Extensional collapse situations. We write $\mathcal{M} \xrightarrow{E} \mathcal{N}$ the fact that $\mathcal{M}, \mathcal{N}$ and the logical relation $E$ determine an extensional collapse situation. When $E$ at ground types is a canonical surjection, clear from the context, we write $\mathcal{M} \twoheadrightarrow \mathcal{N}$.

of the terms (1) and (2) above, when passing from the Scott model to the simple set-theoretic one. They may also “vanish”, since the surjection is partial: this is the case of the Scott-interpretation of (3), the non-total constantly undefined function.

In order to prove that a given pair of models $\mathcal{M}, \mathcal{N}$, determines an extensional collapse situation, we follow the following pattern:

– define a simply typed $\lambda$-calculus $A^\mathcal{N}$, such that $\mathcal{N}$ is fully complete w.r.t. $A^\mathcal{N}$ (i.e. such that all elements of $\mathcal{N}$, at all types, are definable by $A^\mathcal{N}$-terms),
– show that $\mathcal{M}$ is a model of $A^\mathcal{N}$,
– use the fundamental lemma of logical relations (Lemma 6) to exhibit a suitable logical relation, induced by $A^\mathcal{N}$, and conclude.
– an interesting by-product, which is an immediate corollary of the fundamental lemma of logical relations, is that the $A^\mathcal{N}$-theory of $\mathcal{M}$ is included in that of $\mathcal{N}$.

We consider the following models:

– $S$: the full type hierarchy over the set $\{\mathbb{t}, \mathbb{f}\}$. Its elements are “higher order boolean circuits”.
– $C$: the standard model of monotonic$^3$ functions over the partial order $\perp < \mathbb{t}, \mathbb{f}$.
– $E$: the dual of $C$, i.e. the model of monotonic function over the partial order $\mathbb{t}, \mathbb{f} < \top$, modelling unrecoverable errors in the absence of non-termination.
– $L$: the model of monotonic functions over the lattice $\top < \mathbb{t}, \mathbb{f} < \top$, modelling non-termination and errors.

We focus on the extensional collapse situations represented in figure 1. The $\lambda$-calculi used for proving that those are actually extensional collapse situations are $A^S$, whose constants are $\text{true, false}$ and $\text{if}$, and $A^E$, the parallel extension of

$^3$ In the general case, the morphisms of the standard model are the Scott continuous functions. Focusing on finite domains, the monotonic functions and the Scott continuous ones coincide.
finitary PCF, due to Plotkin [12]. It is not necessary to introduce a language \( A^E \)
in order to deal with the model \( E^4 \): we prove instead that \( E \) and \( C \) are logically isomorphic \(^5\), and conclude since extensional collapse situations do compose.

The existence of an extensional collapse situation between two models \( M \) and \( N \) witnesses the fact that the target of the collapse is obtained by forgetting one (or more) computational aspect(s) that are explicitly taken into account in the source.

Let us consider, for instance, the collapse of \( C \) over \( S \): unsurprisingly, at the ground type bool the surjection is the partial identity, undefined on \( \perp \).

At first order types, a monotonic function \( f \in C_\sigma \) is surjected onto the boolean circuit \( c \in S_\sigma \) exactly when it provides the same value as \( c \) on total tuples (i.e. on tuples non containing \( \perp \)). Hence, \( f \) can be considered as an algorithm implementing \( c \).

We have already seen in Example 1 that there exist two implementations of the constantly \( \tt \) function from \( S_{\text{bool}} \) to \( S_{\text{bool}} \), namely the strict and the non-strict constantly \( \tt \) functions from \( C_{\text{bool}} \) to \( C_{\text{bool}} \).

To go a little further, let us consider the \( n \)-ary disjunction \( \text{or}_n \), of type \( \text{bool} \to \ldots \to \text{bool} \to \text{bool} \), yielding the result \( \tt \) whenever at least one of the \( n \) times arguments is \( \tt \) and \( \ff \) if all the arguments are \( \ff \).

They can be implemented in several ways, ranging from the most lazy (and parallel) algorithm yielding the result \( \tt \) whenever at least one of the arguments is \( \tt \), to the most eager one, yielding the result \( \tt \) whenever all the argument are different from \( \perp \), and at least one of them is \( \tt \).

In the case \( n = 2 \), this gives four different implementations of the disjunction, which are usually named, from the laziest to the most eager, parallel-or, left-or, right-or and strict-or.

In the general case, it is not difficult to realise that the algorithms implementing \( \text{or}_n \) form a lattice whose size grows exponentially in \( n \). The laziest algorithm, is the bottom, and the most eager is the top of the lattice.

Summing up, and generalising, we have that for all types \( \sigma \), and all boolean functional \( f \in S_\sigma \), the set of implementations of \( f \) is a sub cpo of \( C_\sigma \), called the totality class of \( f \), which is a lattice.

This “totality collapse” is described in [5].

Hence, the model \( S \) is obtained by collapsing \( C \) via the totality relation; in the same way \( C \) is obtained by collapsing \( L \) via the “error-freedom” relation, which, at ground type, is the partial identity function, undefined on \( \top \).

1.1 Related works

In the literature, a model of the simply typed \( \lambda \)-calculus is called extensional if all its elements, at all types, are invariant with respect to the logical relation defined as the identity at ground types\(^6\).

\(^4\) \( A^C \) could be defined as the dual of \( A^E \) with respect to the inversion of \( \perp \) and \( \top \).

\(^5\) i.e. that they form an extensional collapse situation in either way.

\(^6\) Otherwise stated: \( \forall \sigma, \tau, f, g : \sigma \rightarrow \tau \) if \( \forall x \in \sigma \ f(x) = g(x) \) then \( f = g \).
When a model is not extensional, its extensional collapse is performed by eliminating the non-invariant elements. The result is an extensional model.

This pattern has been followed, for instance, for game models \([1, 9]\), or models obtained by sequentiality relations \([13, 10]\).

Sometimes, the resulting extensional model happens to have been defined and studied independently: those are instances of what we call here extensional collapse situation.

Examples\(^7\) of this kind are:

- sequential algorithms collapsing on strongly stable functions \([6]\).
- the relational model collapsing on Scott-continuous functions (between complete lattices) \([7]\).

Nevertheless, extensional collapse situations as defined in the present paper cover a broader landscape than these extensional collapses. The essential difference is that the extensional collapse of a model is, by definition, unique, whereas different extensional collapse situations may concern a given model (as it is the case of the lattice model \(L\), which collapses over \(\mathcal{C}, \mathcal{E}, \text{and } S\)).

The extensional collapse situation between \(S\) and \(C\) has already been proved in \([5]\), using a different technique. The use of the language \(\Lambda\) and of the fundamental lemma of logical relations makes the proof presented here easier.

Extensional collapses situations, defined in a slightly different way, have been used in \([3]\) for constructing, given two models \(M, N\) of a given applied \(\lambda\)-calculus, a quotient model \(M / N\) whose theory is a super-set of both \(\text{Th}(M)\) and \(\text{Th}(N)\).

2 Type frames and logical relations

Since all the models considered in this paper are extensional, we provide here a very simple definition of type frame, where higher types are interpreted by sets of functions\(^8\).

The set of simple types over a set \(K\) of ground type constants is the smallest set containing \(K\) and closed w.r.t. the operation \(\sigma, \tau \mapsto \sigma \rightarrow \tau\).

**Definition 1.** A type frame \(M\) over a set \(K\) of ground types is a family of sets indexed by simple types over \(K\), such that

\[M_{\sigma \rightarrow \tau} \subseteq \{f \mid f \text{ is a function from } M_\sigma \text{ to } M_\tau\} \]

A type frame is finite if all the sets of the family are finite sets.

**Definition 2.** Given two type frames \(M\) and \(N\) over \(K\), a binary logical relation \(R\) between \(M\) and \(N\) is a family of binary relations \(R_{\sigma} \subseteq M_{\sigma} \times N_{\tau}\), indexed by simple types over \(K\), such that, for all \(\sigma, \tau, f \in M_{\sigma \rightarrow \tau}, g \in N_{\sigma \rightarrow \tau}\):

\[(f, g) \in R_{\sigma \rightarrow \tau} \iff \forall x \in M_\sigma, y \in N_\tau \ [(x, y) \in R_\sigma \Rightarrow (f(x), g(y)) \in R_\tau]\]

\(^7\) It has to be noticed that in these examples the collapse is not a logical relation. It is defined as a categorical notion, independently from the hierarchy of simple types and obviously instantiating to an extensional collapse situation in the sense defined in this paper.

\(^8\) In section 5 we provide the more general definition, encompassing non extensional models.
Definition 3. A binary logical relation $R$ between two type frames $M$ and $N$ over $K$ is a logical surjection if:

- at all types, $R$ is surjective: $\forall \sigma \forall y \in N_{\sigma} \exists x \in M_{\sigma} (x, y) \in R_{\sigma}$
- at all types, $R$ is a partial function: $\forall \sigma \forall x \in M_{\sigma} \forall y, y' \in N_{\sigma} (x, y), (x, y') \in R_{\sigma} \Rightarrow y = y'$

Lemma 1. A binary logical relation between $M$ and $N$ which is a partial function at ground types and surjective at all types is a logical surjection.

Proof. Straightforward, by induction on types. The extensionality of $N$ is required here.

Definition 4. An extensional collapse situation is a triple $M, N, E$, where $M$ and $N$ are type frames over a given set $K$ of ground types and $E$ is a logical surjection from $M$ to $N$.

We note $M \rightarrow^{E} N$ the fact that $M, N, E$ is an extensional collapse situation.

Extensional collapse situations are closed under composition, in the following sense (the proof is straightforward):

Proposition 1. If $M \rightarrow^{E} N$ and $N \rightarrow^{F} P$ then $M \rightarrow^{E \circ F} P$, where $(F \circ E)_{\sigma} = F_{\sigma} \circ E_{\sigma}$.

Hence, the set of type frames over a given set $K$ of ground types is pre-ordered by the relation $N \leq M \iff M \rightarrow^{E} N$, for some $E$. Let $\equiv$ denote the equivalence relation associated to this pre-order.

As a matter of terminology, we call logical isomorphism a logical relation $E$ such that both $E$ and $E^{-1}$ are logical surjection.

2.1 Type frames over bool

In order to define the type frames we are interested in, let us introduce the following notation: if $A$ and $B$ are sets (resp. partially ordered sets), $A \Rightarrow B$ (resp. $A \Rightarrow_{m} B$) denotes the set of all the functions from $A$ to $B$ (resp. the partially ordered set of monotonic functions from $A$ to $B$, ordered pointwise).

We consider four type frames over a single ground type bool: $S, C, E$ and $L$, defined as follows:

- $S_{\text{bool}} = \{\#, \&\}$,
- $S_{\sigma \rightarrow \tau} = S_{\sigma} \Rightarrow S_{\tau}$.
- $C_{\text{bool}} = \{\bot, \#, \&\}$, partially ordered by $\bot < \#, \&$,
- $C_{\sigma \rightarrow \tau} = C_{\sigma} \Rightarrow_{m} C_{\tau}$.
- $E_{\text{bool}} = \{\#, \&, \top\}$, partially ordered by $\#, \&, \top < \top$,
- $E_{\sigma \rightarrow \tau} = E_{\sigma} \Rightarrow_{m} E_{\tau}$.
- $L_{\text{bool}} = \{\bot, \#, \&, \top\}$, partially ordered by $\bot < \#, \&, \top < \top$,
- $L_{\sigma \rightarrow \tau} = L_{\sigma} \Rightarrow_{m} L_{\tau}$.

The type frames $C$ and $E$ are dual. We show that $C \equiv E$ by exhibiting a logical isomorphism.
Proposition 2. The logical relation \( E \) between \( C \) and \( E \) defined by \( E_{\text{bool}} = \{ (\#\#, \#\#), (\#, \#), (\perp, \top) \} \) is a logical isomorphism.

Sketch of the proof. By simultaneous induction on types, prove the following statements:

(i) \( E_{\sigma} \) is a bijection.
(ii) \( \forall (x, y), (x', y') \in E_{\sigma} \ (x \leq x' \iff y' \leq y) \).

3 Applied \( \lambda \)-calculi

The type frames introduced in the previous section are models of simply typed \( \lambda \)-calculi endowed with constants, called applied \( \lambda \)-calculi:

Definition 5. Given a set \( K \) of ground types, an applied \( \lambda \)-calculus \( \Lambda \) over \( K \) is given by a family of typed constants \( C(\Lambda)_\sigma \), indexed by simple types over \( K \).

The terms of the calculus are simply typed \( \lambda \)-terms built by application and \( \lambda \)-abstraction starting from the typed constants and variables.

The operational semantics of an applied \( \lambda \)-calculus is specified by a set of \( \delta \)-rules, stipulating the behaviour of the constants, and by the \( \beta \)-rule.

As a matter of notation, we will write \( \Lambda_\sigma \) (resp. \( \Lambda_0^\sigma \)) for the set of terms (resp. closed terms) of type \( \sigma \).

The interpretation of a closed term \( P \) will be noted simply \( \langle P \rangle_\sigma \) or \( \langle P \rangle \), when non ambiguous.

Two conditions have to be satisfied for a type frame \( M \) to be a model of a given applied \( \lambda \)-calculus:

1. In the fourth item of the definition above, one has that the function \( d \in M_{\sigma \rightarrow \tau} \) is an element of \( M_{\sigma \rightarrow \tau} \), for the appropriate types \( \tau \).
2. The map \( C \) is sound: for any \( \delta \)-rule \( P \rightarrow^\delta Q \), one has \( \langle P \rangle \approx \langle Q \rangle \).

When these conditions are satisfied, the model is such that \( P \rightarrow^* Q \Rightarrow \langle P \rangle_\sigma = \langle Q \rangle_\sigma \), where the rewriting relation \( \rightarrow \) is the contextual closure of \( \rightarrow^\delta \cup \delta \).

Note that, for the type frames \( S, C, E \) and \( L \), the first condition is always satisfied, since for all of them \( [\sigma \rightarrow \tau] \) is the exponential object \( [\tau]^{[\sigma]} \) in some Cartesian closed category (actually there exists a ccc which is an ambient category for all those type frames, namely the category of finite partial orders and monotone functions). Hence, in order to show that a given type frame among
$S, C, E$ and $\mathcal{L}$ is a model of a given applied $\lambda$-calculus, it is sufficient to provide a sound interpretation of the constants of the language.

Among the models of a given applied $\lambda$-calculus, the fully complete ones are those whose elements are definable (in general, one asks for the definability of finite elements. We skip this condition here since we focus on finite type frames).

**Definition 6.** A model $M$ of an applied $\lambda$-calculus $\Lambda$ is fully complete if for all $\sigma$ and for all $d \in M_{\sigma}$ there exists a closed $\Lambda$-term $D$ of type $\sigma$ such that $[D]^{M} = d$.

We define now the applied $\lambda$-calculi that will be used in the sequel to prove some extensional collapse situations.

We call these calculi $\Lambda^S$ and $\Lambda^C$ respectively, to emphasize the full completeness of the corresponding type frames.

### 3.1 The basic boolean calculus

Here there is the definition of the constants of $\Lambda^S$ and of their operational semantics:

**Definition 7.**
- $C(\Lambda^S)_{\text{bool}} = \{ \text{true}, \text{false} \}$
- $C(\Lambda^S)_{\text{bool} \rightarrow \text{bool} \rightarrow \text{bool} \rightarrow \text{bool}} = \{ \text{if} \}$
- $\text{if true } M \ N \rightarrow^{S} M$, $\text{if false } M \ N \rightarrow^{S} N$

**Lemma 2.** $S$ is a fully complete model of $\Lambda^S$.

*Sketch of the proof.* First of all, the interpretation mapping the ground constants on the corresponding boolean values and such that $[\text{if}]^{S} d e f = \begin{cases} e & \text{if } d = \text{true} \\ f & \text{if } d = \text{false} \end{cases}$ is clearly sound, hence $S$ is a model of $\Lambda^S$.

Concerning full completeness: consider an element $f = \{ (a_1, b_1), \ldots, (a_n, b_n) \} \in S_{\sigma \rightarrow \tau}$. If the $a_i$, the $b_j$ and the equality predicate on $S_{\sigma}$ were $\Lambda^S$-definable, then it would be easy to write down a $\Lambda^S$-term defining $f$, as a sequence of nested if.

On the other hand, one can $\Lambda^S$-define the equality predicate on $S_{\sigma \rightarrow \tau}$ if one is able to define the elements of $S_{\sigma}$ and the equality predicate on $S_{\tau}$.

Hence, the proof consists in a straightforward simultaneous induction on types, for the two following properties:

- $\text{DEF}(\sigma)$: all the elements of $S_{\sigma}$ are $\Lambda^S$-definable.
- $\text{EQ}(\sigma)$: the equality predicate on $S_{\sigma}$ is $\Lambda^S$-definable.

By the way, $\text{EQ}([\text{bool}])$ is proved by the term $\lambda x : \text{bool} \ y : \text{bool}. \text{if } x \ (\text{if } y \ \text{true} \ \text{false}) \ (\text{if } y \ \text{false} \ \text{true})$.

**Lemma 3.** $C$ and $E$ are models of $\Lambda^S$.

*Proof.* The ground constants $\text{true}$ and $\text{false}$ are interpreted by the corresponding booleans, in both models. Here there is the interpretation of the constant $\text{if}$ in the two models:
3.2 Adding non-termination

Non-termination can be added to the basic calculus by means of fixpoint combinators, for instance. Nevertheless, a single new constant $\Omega : \text{bool}$, whose intended interpretation is the undefined boolean value $\bot$ is enough. In fact, fixpoint combinators do not add expressive power in the case of finite ground types, apart from the possibility of divergence. In order to achieve the full completeness of $\mathcal{C}$, we add a parallel-or constant to the language, following Plotkin [12].

$A^C$ is the extension of $A^S$ defined as follows:

Definition 8. $\forall \sigma \ C(A^S)_\sigma \subseteq C(A^C)_\sigma$ and:

- $\Omega \in C(A^C)_\text{bool}$
- $C(A^C)_\text{bool} \longrightarrow \text{bool} = \{ \text{por} \}$
- $\text{por} \text{true} \ M \ \rightarrow^b \text{true, por} \ M \text{true} \rightarrow^b \text{true}$
- $\text{por} \text{false} \text{false} \rightarrow^b \text{false}$

Lemma 4. $\mathcal{C}$ is a fully complete model of $A^C$.

Proof. This is a corollary of Plotkin’s proof of full abstraction of the Scott model of parallel PCF [12].

Lemma 5. $\mathcal{L}$ is a model of $A^C$.

Proof. The ground constants $\text{true}$ and $\text{false}$ and $\Omega$ are interpreted by $\bot$, $\text{ff}$ and $\bot$ respectively. Here are the interpretations of $\text{if}$ and $\text{por}$ in $\mathcal{L}$:

\[
[\text{if}]^C_{d \ e \ f} = \begin{cases} 
\bot & \text{if } d = \bot \\
 e & \text{if } d = \# \\
 f & \text{if } d = \text{ff} \\
 \top & \text{if } d = \top 
\end{cases} \quad [\text{por}]^C_{d \ e} = \{ \# \mid d \geq \# \text{ or } e \geq \# \} \lor \{ \text{ff} \mid d \geq \text{ff} \text{ and } e \geq \text{ff} \} 
\]

Otherwise stated, the interpretation of $\text{por}$ in $\mathcal{L}$ is given by the following truth table:

\[
\begin{array}{cccc}
\text{por} & \bot & \# & \top \\
\bot & \bot & \bot & \# \\
\# & \bot & \# & \top \\
\text{ff} & \bot & \# & \top \\
\text{ff} & \text{ff} & \text{ff} & \text{ff} \\
\# & \# & \# & \# \\
\top & \# & \# & \# \\
\end{array}
\]

This interpretation validates the $\delta$-rules concerning $\text{if}$ and $\text{por}$. 
4 Extensional Collapse Situations

In this section, we prove the six extensional collapse situations exhibited in fig. 1. An effective way of providing an extensional collapse situation $\mathcal{M} \rightarrow^E \mathcal{N}$, is to define an applied $\lambda$-calculus $\Lambda^N$ such that $\mathcal{N}$ is a fully complete model of $\Lambda^N$, and $\mathcal{M}$ is a model of $\Lambda^N$; then full completeness may be used to exhibit a suitable logical surjection. The key fact is expressed by the fundamental lemma of logical relations [11], below:

**Lemma 6.** Let $\mathcal{M}, \mathcal{N}$ be models of an applied $\lambda$-calculus $\Lambda$, and let $E$ be a logical relation between $\mathcal{M}$ and $\mathcal{N}$, such that for all constants $c : \sigma$ of $\Lambda$, $([c]^\mathcal{M}, [c]^\mathcal{N}) \in E_\sigma$. Then, for all type $\sigma$ and for all $P \in \Lambda^0_\sigma$, $([P]^\mathcal{M}, [P]^\mathcal{N}) \in E_\sigma$.

**Corollary 1.** Let $\mathcal{M}, \mathcal{N}$ be type frames over $K$, and suppose that for all ground types $k \in K$ there exists a partial surjective function $E_k \subseteq \mathcal{M}_k \times \mathcal{N}_k$. If there exists an applied $\lambda$-calculus $\Lambda$, such that, for all constants $c : \sigma$ in $\Lambda$, $([c]^\mathcal{M}, [c]^\mathcal{N}) \in E_\sigma$, and such that $\mathcal{N}$ is fully complete for $\Lambda$. Then $\mathcal{M} \rightarrow^E \mathcal{N}$.

**Proof.** We have to prove that $E_\sigma$ is a partial surjective function, for all $\sigma$. For ground types, the statement holds by hypothesis. By using Lemma 1 we are left with proving that $E$ is surjective at higher types.

Given $e \in \mathcal{N}_\sigma$, let $P \in \Lambda^0_\sigma$ be such that $[P]^\mathcal{N} = e$.

By Lemma 6 we have that $([P]^\mathcal{M}, [P]^\mathcal{N}) \in E_\sigma$, and we are done.

Given an applied $\lambda$-calculus $\Lambda$ and one of its models $\mathcal{M}$, let $Th^\Lambda(\mathcal{M}) = \{ P = Q \mid P, Q \in \Lambda^0 \text{ and } [P]^\mathcal{M} = [Q]^\mathcal{M} \}$.

**Corollary 2.** Let $\mathcal{M}$ and $\mathcal{N}$ be models of an applied $\lambda$-calculus $\Lambda$, and $\mathcal{M} \rightarrow^E \mathcal{N}$ be an extensional collapse situation. If, for all constants $c : \sigma$ of $\Lambda$, $([c]^\mathcal{M}, [c]^\mathcal{N}) \in E_\sigma$, then $Th^\Lambda(\mathcal{M}) \subseteq Th^\Lambda(\mathcal{N})$.

**Proof.** Let us suppose that $[P]^\mathcal{M} = [Q]^\mathcal{M}$, for two given closed $\Lambda$-terms $P$ and $Q$. By lemma 6 we have that $([P]^\mathcal{M}, [P]^\mathcal{N}), ([Q]^\mathcal{M}, [Q]^\mathcal{N}) \in E_\sigma$. Since $E_\sigma$ is a function, we conclude that $[P]^\mathcal{N} = [Q]^\mathcal{N}$.

Here is our main result:

**Proposition 3.** The following extensional collapse situations do hold:

$C \rightarrow S$, $E \rightarrow S$, $C \rightarrow E$, $E \rightarrow C$, $L \rightarrow C$, $L \rightarrow E$.

**Proof.**

- $C \rightarrow S$ and $E \rightarrow S$ follow from Corollary 1, where the partial surjection at ground type is $\# \mapsto \#$, $ff \mapsto ff$ and the applied $\lambda$-calculus is $\Lambda^S$, and Lemmata 2, 3.
- $C \rightarrow E$ and $E \rightarrow C$ follow from Proposition 2
- $L \rightarrow C$ follows from Corollary 1, where the partial surjection at ground type is $\bot \mapsto \bot$, $\# \mapsto \#$, $ff \mapsto ff$ and the applied $\lambda$-calculus is $\Lambda^E$, and Lemmata 4, 5.
- \( L \rightarrow \mathcal{E} \) follows from the two items above and Proposition 1.

Note that, by composition of extensional collapse situations, we obtain \( L \rightarrow C \). Actually, there are four ways of obtaining it, corresponding to the different paths from \( L \) to \( S \) in Figure 1. It is easy to see that these paths give rise to the same partial surjection.

By Corollary 2, the proposition above yields in particular:

**Corollary 3.**

- \( \text{Th}^{A^C}(\mathcal{C}) \subseteq \text{Th}^{A^S}(\mathcal{S}) \)
- \( \text{Th}^{A^C}(\mathcal{L}) \subseteq \text{Th}^{A^C}(\mathcal{C}) \)

The examples presented in Section 1 show that the inclusions above are strict.

## 5 Conclusion and further work

We have provided a general definition of “inclusion” of models of higher order, functional computations via the notion of extensional collapse situation, and shown this notion at work on some simple models over bool.

As suggested by the title, this work should be considered as a first step settling the ground for the study of extensional collapse situations on more complicated models. As a matter of fact, the definitions of section 2 have to be generalised a bit to encompass non extensional models. In particular, if \( \mathcal{C} \) is a Cartesian closed category without enough points, the corresponding type frame \( \mathcal{C}_\sigma \) is defined by:

- \( C_{\text{bool}} = \mathcal{C}(1, C_{\text{bool}}) \) for a suitable object \( C_{\text{bool}} \)
- \( C_{\sigma \rightarrow \tau} = \mathcal{C}(1, \mathcal{C}_\sigma \Rightarrow \mathcal{C}_\tau) \)

and, given \( f \in C_{\sigma \rightarrow \tau} \) and \( x \in \mathcal{C}_\sigma \), one defines \( f(x) = \text{ev}_\circ \langle f, x \rangle \).

Then, the fundamental lemma 6 holds and the construction of extensional collapses situations via applied \( \lambda \)-calculi goes through. This generalisation of logical relations for categories without enough points is described for instance in [2], section 4.5.

Let us introduce a direction for further work via an example, referring to the ones showed in the introduction. The terms:

- \( \lambda x: \text{bool}. \ x \)
- \( \lambda x: \text{bool}. \ \text{if} \ x \ \text{then} \ x \ \text{else} \ x \)

get the same interpretation in all the models considered in the present paper. They get different interpretation, for instance\(^9\), in the model \( \mathcal{R} \) of set and relations [8], endowed with the finite multiset exponential (see for instance [4] for a direct description of that model as a Cartesian closed category).

Using Ehrhard’s result on the extension collapse of \( \mathcal{R} \) over \( \mathcal{L} \) [7], we should obtain the following chain of extensional collapse situations:

\( ^9 \) Actually, these terms get different interpretations in all models for which *linearity* is an extensional property. This character of the model is often termed “resource sensitivity”.

An interesting sub-problem is the definition of an extension of (finitary) PCF w.r.t. which \( L \) is fully complete.

What comes after? More intensional models, like game models, could collapse over \( R \). Also, where is the place of models based on stable orderings, as the stable or strongly stable ones, in this hierarchy?

We already know some partial answer (for instance, the fact that Berry and Curien’s sequential algorithms collapse over the model of strongly stable functions [6]).

Quite a number of different models of higher order, functional computation exist in literature.

Having a better overview of the poset of extensional collapse situations relating these models may contribute to a better understanding of the whole picture.

References