Subtyping + Extensionality: Confluence of $\beta\eta$ Top reduction in $F_\leq$

Pierre-Louis Curien\(^1\), Giorgio Ghelli\(^2\)

Abstract: We contribute to the syntactic study of $F_\leq$, a variant of second order $\lambda$-calculus $F$ which appears as a paradigmatic kernel language for polymorphism and subtyping. The type system of $F_\leq$ has a maximum type Top and bounded quantification. We endow this language with the familiar $\beta$-rules (for terms and types), to which we add extensionality rules: the $\eta$-rules (for terms and types), and a rule (top) which equates all terms of type Top. These rules are suggested by the axiomatization of cartesian closed categories. We show that this theory $\beta\eta$ Top$_\leq$ is decidable, by exhibiting an effectively weakly normalizing and confluent rewriting system for it. Our proof of confluence relies on the confluence of a corresponding system on $F_1$ (the extension of $F$ with a terminal type), and follows a general pattern that we investigate for itself in a separate paper.

After giving some background on the language $F_\leq$ (section 1), we make observations on the confluence problem of $\beta\eta$ in various typed $\lambda$-calculi (section 2). We discuss some difficulties arising from the multiplicity of typing proofs in $F_\leq$ (section 3), which lead us to study the corresponding problem in an explicit calculus $eF_\leq$, whose terms codify typing proofs in $F_\leq$ (section 4). We add the (top) rule which equates all terms of type Top (section 5). In the rest of the paper (sections 6 through 10) we state our confluence results (theorem 1 in section 9, and theorem 2 in section 10), and we offer a technical survey of the constructions which lead us to them. We conclude in section 11.

1 $F_\leq$ background

$F_\leq$ is a basic language expressing structural subtyping and polymorphism. It allows for an encoding of most familiar structures with subtyping such as records and variants, and is a kernel for the language Quest developed by Luca Cardelli [Ca]. It is essentially the language Fun of [CaWe] (unlike $F_\leq$, Fun has recursive types and values, and has records, variants and existential types as primitives). $F_\leq$ was introduced in [CG1] (see also [BrLo] and [CaLo]). We briefly recall the syntax of $F_\leq$, and hint at a few encodings. The reader will find more material in [CaLo,GheTh,CaMaMiSce].

\(^1\) LIENS (CNRS), 45 rue d'Ulm, 75230 Paris Cedex 05, France. This work was carried on with the partial support of E.E.C., Esprit Basic Research Action 3003 CLICS.
\(^2\) Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56100, Pisa, Italy, ghelli@dipisa.di.unipi.it. This work was carried on with the partial support of E.E.C., Esprit Basic Research Action 3070 FIDE and of Italian C.N.R., P.F.I. "Sistemi informatici e calcolo parallelo".
The types and expressions of $F_\preceq$ are defined as follows:

$$
A ::= t \mid \text{Top} \mid A \rightarrow A \mid \forall t \preceq A. A \\
\alpha ::= x \mid \text{top} \mid \lambda x:A. \alpha \mid \alpha(a) \mid \forall t \preceq A. \alpha \mid a\{A\}
$$

In the second order abstraction $\forall t \preceq A. \alpha$, the bound $A$ restricts the types which can be accepted as parameters to be subtypes of $A$. The type $\text{Top}$ is a supertype of all types, so that unbounded second order lambda abstraction can be recovered as $\forall t \preceq \text{Top}. \alpha$. The term $\text{top}$ is the (unique) term of type $\text{Top}$. The typing rules are given in appendix A.

Here are a few examples of subtypings. In $F_\preceq$ the known encoding of the type of booleans in second order $\Lambda$-calculus, $TF \equiv \forall t.t \rightarrow t \rightarrow t$, has four subtypes:

- $TF \equiv \forall t \preceq \text{Top}. t \rightarrow t \rightarrow t$ (the booleans)
- $F \equiv \forall t \preceq \text{Top}. t \rightarrow t \rightarrow t$ (just false)
- $T \equiv \forall t \preceq \text{Top}. t \rightarrow t \rightarrow t$ (just true)
- $\bot \equiv \forall t \preceq \text{Top}. t \rightarrow t$ (an empty type)

One can prove the following inclusions: $\bot \preceq \bot$, $\bot \preceq F$, $T \preceq TF$ and $F \preceq TF$. Moreover, there is no term of type $\bot$, and only one $\beta$ normal form $\text{true}_T = \forall t \preceq \text{Top}. \lambda x:t. \lambda y: \text{Top}. x$ of type $T$. In $TF$ we find the following normal forms: $\text{true}_T$ again, and

$$
\text{true}_{TF} = \forall t \preceq \text{Top}. \lambda x:t. \lambda y:t. x
$$

(and similarly $\text{false}_F$ and $\text{false}_{TF}$). We shall see in section 3 that $\text{true}_F$ and $\text{true}_{TF}$, $\text{false}_F$ and $\text{false}_{TF}$ are provably $\beta\eta$ equal. So the subtypes of $TF$ correspond exactly to the subsets of \{true, false\}.

We now briefly hint at the encoding of records. The essence of record subtyping is the possibility of obtaining subtypes by adding fields. This is captured by Cardelli’s $n$-ary tuples [CaLo], which satisfy the following subtyping rule:

$$
A_1 \leq A'_1 \Rightarrow A_1 \otimes \ldots \otimes A_n \otimes B \leq A'_1 \otimes \ldots \otimes A'_n .
$$

A tuple can be encoded in $F_\preceq$ as the right associative product of its component types and of $\text{Top}$. For example $A_1 \otimes \ldots \otimes A_n \otimes B$ and $A'_1 \otimes \ldots \otimes A'_n$ are encoded respectively as

$$(A_1 \times \ldots \times (A_n \times (B \times \text{Top}))) \ldots) \text{ and } (A'_1 \times \ldots \times (A'_n \times (\text{Top}))) \ldots$$

and the inequality follows from $A_1 \preceq A'_1$ and $B \times \text{Top} \preceq \text{Top}$. This encoding of tuples can be easily extended to records [CaLo].

In a previous paper [CG1], we initiated a study of the syntactic theory of $F_\preceq$. We addressed a coherence problem: in presence of subtyping, there may be several proofs establishing that a given expression has a given type. Similarly there may be several proofs of a subtyping judgement $A \preceq B$. In [CG1] we have exhibited a complete equational axiomatization of the following equivalence relation between typing proofs: two proofs are equivalent when they prove the same typing or subtyping judgement (see proposition 3 of this paper). To prove the