Computer Science at Kent

Yet Another Normalisation Proof for Martin-Löf's Logical Framework — Terms with correct arities are strongly normalising

Yong Luo

Technical Report No. 6 - 05
October 2005

Copyright © 2005 University of Kent

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent, CT2 7NF, UK

Yet Another Normalisation Proof for Martin-Löf's Logical Framework

— Terms with correct arities are strongly normalising

Yong Luo

Computing Laboratory, University of Kent, Canterbury, UK
Email: Y.Luo@kent.ac.uk

Abstract In this paper, we prove the strong normalisation for Martin-Löf's Logical Framework, and suggest that "correct arity", a condition weaker than well-typedness, will also guarantee the strong normalisation.

1 Introduction

The normalisation proofs for dependently typed systems are known to be notoriously difficult. For example, if we have a task to prove strong normalisation for Martin-Löf's Logical Framework (MLF) (in the Appendix), and if we use typed operational semantics as in [Gog94], the proof would be more than one hundred pages long. When a proof is long and complicated, it is likely found to contain mistakes and bugs [Coq85,CG90,Alt94]. This paper presents an elegant and comprehensible proof of strong normalisation for MLF.

We often associate well-typedness with strong normalisation in type systems. But this paper suggests that well-typedness may have little to do with strong normalisation in essence, and proves that terms with correct arities are strongly normalising. The condition of "correct arity" is weaker than that of well-typedness (*i.e.* well-typed terms have correct arities). This paper will also demonstrate the difference between types and arities when we extend MLF with some inductive data types and their computation rules. New reduction rules will not increase the set of terms with correct arities, but they will usually increase the set of well-typed terms. One of the reasons is that there are reductions inside types (*i.e.* one type can be reduced to another type) in a dependently typed system but there is no reduction for arities.

Our goal is to prove the strong normalisation w.r.t. β and η-reduction. But it is very difficult to prove it directly. An important technique in the paper is that, we extend the definition of terms and kinds, and introduce a new reduction rule β_2 for kinds. Then, we prove a stronger and more general property, that is, strong normalisation w.r.t. β, η and β_2-reduction. In this way, the proof becomes easier although the property is stronger. Without the β_2-reduction, the proof of soundness in Section 4 is impossible to go through.

In Section 2, we give some basic definitions that are used throughout the paper. In Section 3, the inference rules of arities are formally presented. In Section 4, we give more definitions such as saturated sets, and prove the strong normalisation for the arity system. In Section 5, the computation rules for the type of dependent pairs and finite types and simple computation rules for universes are introduced. The strong normalisation for a dependently typed system is proved by the commutation property between these rules and β-reduction. The conclusions and future work are discussed in the last section.

Related work Logical frameworks arise because one wants to create a single framework, which is a kind of meta-logic or universal logic. The Edinburgh Logical Framework [HHP87,HHP92] presents logics by a *judgements-as-types* principle, which can be regarded as the meta-theoretical analogue of the well-known *propositions-as-types* principles [CF58,dB80,How80]. Martin-Löf's logical framework [ML84,NPS90] has been developed by Martin-Löf to present his intensional type theory. In UTT [Luo94], Luo proposed a typed version of Martin-Löf's logical framework, in which untyped functional operations of the form $(x)k$ are replaced by typed $[x : K]k$.

There are many normalisation proofs for simply typed systems and dependently typed systems in literature [Bar92,Luo90,Alt93] [MW96,Gog94] [Geu93,Wer92]. The techniques employed in this paper such as the interpretation of arities and saturated sets are inspired by and closely related to the proof for simply typed calculus in [Bar92]. The concept of arity is well-known in mathematics and it is often defined as the maximum number of arguments that a function can have. But in this paper, the definition of arity and the concept of "correct arity" are different. The complexity of the normalisation proof for MLF is dramatically decreased because of this concept and other techniques such as a new case of kinds and the corresponding β_2-reduction. The commutation property was also studied in literature such as [Bar84,Cos96]. The properties of Church-Rosser and strong normalisation for finite types in simply typed systems are also studied in [SC04].

2 Basic definitions

In this section, we give some basic definitions that will be used later, and give the redexes and the corresponding reduction rules.

Definition 1. ***(Terms and Kinds)***

- *Terms*
  1. *a variable is a term,*
  2. *$\lambda x : K.M$ is a term if x is a variable, K is a kind and M is a term,*
  3. *MN is a term if M and N are terms.*
- *Kinds*
  1. *Type is a kind,*
  2. *$El(M)$ is a kind if M is a term,*
  3. *$(x : K_1)K_2$ is a kind if K_1 and K_2 are kinds,*
  4. *KN is a kind if K is a kind and N is a term.*

Remark 1. Terms and kinds are mutually and recursively defined. This definition allows more terms and kinds than that of MLF since the forth case for the definition of kinds is not included in MLF (see Appendix for details).

Notation: Following the tradition, Λ denotes the set of all terms and Π the set of all kinds. We sometimes write $f(a)$ for fa, $f(a,b)$ for $(fa)b$ and so on. $[N/x]M$ stands for the expression obtained from M by substituting N for the free occurrences of variable x in M. $FV(M)$ is the set of free variables in M.

Redexes and reduction rules

There are three different forms of redexes: $(\lambda x : K.M)N$, $((x : K_1)K_2)N$ and $\lambda x : K.Mx$ when $x \notin FV(M)$. The reduction rules for these redexes are the following.

$$(\lambda x : K.M)N \longrightarrow_{\beta} [N/x]M$$

$$((x : K_1)K_2)N \longrightarrow_{\beta_2} [N/x]K_2$$

$$\lambda x : K.Mx \longrightarrow_{\eta} M \qquad x \notin FV(M)$$

Remark 2. The second rule $\longrightarrow_{\beta_2}$ is new and is not included in MLF. This rule will make the soundness proof go through easily although the property is stronger and more general.

Notation: $\longrightarrow_R$ represents one-step R-reduction, precisely, $M \longrightarrow_R N$ if a sub-term P of M is a R-redex and N is obtained by replacing P by the result after applying the reduction rule R. $M \twoheadrightarrow_R N$ means there is 0 or more but finite steps of R-reduction from M to N. $M \twoheadrightarrow_R^+ N$ means there is at least one but finite steps of R-reduction from M to N.

Definition 2. ***(Arities)***

- *Zero is an arity,*
- (a_1, a_2) *is an arity if* a_1 *and* a_2 *are arities.*

Notation: Ω denotes the set of all arities.

Remark 3. The concept of arity in mathematics is often defined as the maximum number of arguments that a function can have. For example, the arity of the function "addition" is 2, but in this paper, the arity of "addition" is $(Zero, (Zero, Zero))$.

3 Inference rules

In this section, we formally present the inference rules of arities.

The judgement form will be the following form,

$$A \vdash M : a$$

where $A \equiv < x_1 : a_1, ..., x_n : a_n >$ is a finite sequence of $x_i : a_i$, x_i is a variable and a_i is an arity; M is a term or kind; and a is an arity. We shall read this judgement like "under the context A, the term or kind M has arity a".

Notation For a context $A \equiv x_1 : a_1, ..., x_n : a_n$, $FV(A)$ represents the set $\{x_1, ..., x_n\}$.

All of the inference rules of arities are in Figure 1.

Definition 3. *We say that a term or kind M has* ***a correct arity*** *if $A \vdash M : a$ is derivable for some A and a.*

Remark 4. We have the following remarks:

Contexts:

$$\frac{}{<>\ \ valid} \qquad \frac{A\ \ valid \quad x \notin FV(A) \quad a \in \Omega}{A, x : a}$$

Inference rules for kinds:

$$\frac{A\ \ valid}{A \vdash Type : Zero} \qquad \frac{A \vdash M : Zero}{A \vdash El(M) : Zero}$$

$$\frac{A \vdash K_1 : a_1 \quad A, x : a_1 \vdash K_2 : a_2}{A \vdash (x : K_1)K_2 : (a_1, a_2)} \qquad \frac{A \vdash K : (a_1, a_2) \quad A \vdash N : a_1}{A \vdash KN : a_2}$$

Inference rules for terms:

$$\frac{A, x : a, A'\ \ valid}{A, x : a, A' \vdash x : a}$$

$$\frac{A \vdash K : a_1 \quad A, x : a_1 \vdash M : a_2}{A \vdash \lambda x : K.M : (a_1, a_2)} \qquad \frac{A \vdash M : (a_1, a_2) \quad A \vdash N : a_1}{A \vdash MN : a_2}$$

Figure 1. Inference rules of arities

- A well-typed term has a correct arity (a proof will be given later), but a term which has a correct arity is not necessarily well-typed. For instance, under the context

  $$A : Type, B : Type, C : Type, f : (x : A)C, b : B$$

  the term $f(b)$ is not well-typed, but it has a correct arity $Zero$ under the following context

  $$A : Zero, B : Zero, C : Zero, f : (Zero, Zero), b : Zero$$

  Another example with dependent type is that, under the context

  $$A : Type, B : (x : A)Type, f : (x : A)(y : B(x))Type,\\ x_1 : A, x_2 : A, b : B(x_2)$$

  the term $f(x_1, b)$ is not well-typed, but it has a correct arity $Zero$ in the following context

  $$A : Zero, B : (Zero, Zero), f : (Zero, (Zero, Zero)),\\ x_1 : Zero, x_2 : Zero, b : Zero$$

- For any judgement $A \vdash M : a$, M must be either a kind or a term. A derivation such as $\frac{A \vdash Type:Zero}{A \vdash El(Type):Zero}$ is not possible, because $El(Type)$ is neither a term nor a kind.

Lemma 1. *If both $A \vdash M : a$ and $A \vdash M : b$ are derivable then a and b are syntactically the same ($a \equiv b$). And $A \vdash MM : a$ is not derivable for any A, M and a.*

Proof. By induction on the derivations of $A \vdash M : a$ and $A \vdash M : b$.

Remark 5. One may recall that the non-terminating example $\omega\omega$ where $\omega \equiv \lambda x.xx$. It is impossible that ω is well-typed in a simply typed calculus [Bar92]. By Lemma 1, it is also impossible to have a correct arity for ω.

4 Normalisation proof

In this section, we give more definitions such as saturated sets to prove the strong normalisation for the arity system.

Definition 4. ***(Interpretation of arities)***

- $SN^{\Lambda} =_{df} \{M \in \Lambda \mid M \, is \, strongly \, normalising\}$.
- $SN^{\Pi} =_{df} \{M \in \Pi \mid M \, is \, strongly \, normalising\}$.
- $[\![Zero]\!]^{\Lambda} =_{df} SN^{\Lambda}$.
- $[\![Zero]\!]^{\Pi} =_{df} SN^{\Pi}$.
- $[\![(a_1, a_2)]\!]^{\Lambda} =_{df} \{M \in \Lambda \mid \forall N \in [\![a_1]\!]^{\Lambda},\ MN \in [\![a_2]\!]^{\Lambda}\}$.
- $[\![(a_1, a_2)]\!]^{\Pi} =_{df} \{K \in \Pi \mid \forall N \in [\![a_1]\!]^{\Lambda},\ KN \in [\![a_2]\!]^{\Pi}\}$.

Remark 6. $[\![a]\!]^{\Lambda}$ is a set of terms, while $[\![a]\!]^{\Pi}$ is a set of kinds for any arity a.

Notations: We shall write $\overline{R}$ for $R_1, R_2, ..., R_n$ for some $n \geq 0$, and $M\overline{R}$ for $(...((MR_1)R_2)...R_n)$.

Definition 5. ***(Saturated sets)***

- *A subset $X \subseteq SN^{\Lambda}$ is called saturated if*
  1. *$\forall \overline{R} \in SN^{\Lambda}$, $x\overline{R} \in X$ where x is any term variable,*
  2. *$\forall \overline{R} \in SN^{\Lambda}$, $\forall Q \in SN^{\Lambda}$ and $\forall K \in SN^{\Pi}$,*

$$([Q/x]P)\overline{R} \in X \Longrightarrow (\lambda x : K.P)Q\overline{R} \in X$$

- *A subset* $Y \subseteq SN^{\Pi}$ *is called saturated if* $\forall \overline{R} \in SN^{\Lambda}$, $\forall N \in SN^{\Lambda}$ *and* $\forall K_1 \in SN^{\Pi}$,

$$([N/x]K_2)\overline{R} \in Y \Longrightarrow ((x:K_1)K_2)N\overline{R} \in Y$$

- $SAT^{\Lambda} =_{df} \{X \subseteq SN^{\Lambda} \mid X\, is\, saturated\}$
- $SAT^{\Pi} =_{df} \{Y \subseteq SN^{\Pi} \mid Y\, is\, saturated\}$

Lemma 2. ***(Arities and saturated sets)***

- $SN^{\Lambda} \in SAT^{\Lambda}$ *and* $SN^{\Pi} \in SAT^{\Pi}$.
- $a \in \Omega \Longrightarrow [\![a]\!]^{\Lambda} \in SAT^{\Lambda}$ *and* $[\![a]\!]^{\Pi} \in SAT^{\Pi}$.

Proof. By the definition of saturated sets and by induction on arities.

- Let's prove $SN^{\Lambda} \in SAT^{\Lambda}$ first. We have $SN^{\Lambda} \subseteq SN^{\Lambda}$ and $x\overline{R} \in SN^{\Lambda}$ if $\overline{R} \in SN^{\Lambda}$. Now we need to prove for $Q, \overline{R} \in SN^{\Lambda}$ and $K \in SN^{\Pi}$,

$$([Q/x]P)\overline{R} \in SN^{\Lambda} \Longrightarrow (\lambda x : K.P)Q\overline{R} \in SN^{\Lambda}$$

Since $([Q/x]P)\overline{R} \in SN^{\Lambda}$, we have $P \in SN^{\Lambda}$ and after any finitely many steps reducing inside P, Q and $\overline{R}$, $([Q'/x]P')\overline{R'} \in SN^{\Lambda}$ with $P \twoheadrightarrow_{\beta\eta} P'$, $Q \twoheadrightarrow_{\beta\eta} Q'$ and $\overline{R} \twoheadrightarrow_{\beta\eta} \overline{R'}$.
From $(\lambda x : K.P)Q\overline{R}$, after any finitely many steps reducing inside P, Q, $\overline{R}$ and K, and we get $(\lambda x : K'.P')Q'\overline{R'}$. From here, we may have two choices.

  · $(\lambda x : K'.P')Q'\overline{R'} \longrightarrow_{\beta} ([Q'/x]P')\overline{R'}$
  · $P' \equiv Fx$ and $x \notin FV(F)$ and

$$(\lambda x : K'.P')Q'\overline{R'} \longrightarrow_{\eta} FQ'\overline{R'} \equiv ([Q'/x]P')\overline{R'}$$

For both cases, because $([Q'/x]P')\overline{R'} \in SN^{\Lambda}$, we have $(\lambda x : K.P)Q\overline{R} \in SN^{\Lambda}$.
- The proof of $SN^{\Pi} \in SAT^{\Pi}$ is similar to that of $SN^{\Lambda} \in SAT^{\Lambda}$.
- Now, let's prove $[\![a]\!]^{\Lambda} \in SAT^{\Lambda}$ by induction on a. The base case (*i.e.* $[\![Zero]\!]^{\Lambda} = SN^{\Lambda} \in SAT^{\Lambda}$) has been proved. So we only need to prove $[\![(a_1, a_2)]\!]^{\Lambda} \in SAT^{\Lambda}$. By induction hypothesis, we have $[\![a_1]\!]^{\Lambda} \in SAT^{\Lambda}$ and $[\![a_2]\!]^{\Lambda} \in SAT^{\Lambda}$.
Then we have $x \in [\![a_1]\!]^{\Lambda}$ for all variable x. Therefore

$$\begin{aligned} F \in [\![(a_1, a_2)]\!]^{\Lambda} &\Longrightarrow Fx \in [\![a_2]\!]^{\Lambda} \\ &\Longrightarrow Fx \in SN^{\Lambda} \\ &\Longrightarrow F \in SN^{\Lambda} \end{aligned}$$

So, we have $[\![(a_1, a_2)]\!]^{\Lambda} \subseteq SN^{\Lambda}$.
Now, we need to prove that for any variable x and $\forall \overline{R} \in SN^{\Lambda}$, we have $x\overline{R} \in [\![(a_1, a_2)]\!]^{\Lambda}$. This means

$$\forall N \in [\![a_1]\!]^{\Lambda} \quad x\overline{R}N \in [\![a_2]\!]^{\Lambda}$$

which is true since $[\![a_1]\!]^{\Lambda} \subseteq SN^{\Lambda}$ and $[\![a_2]\!]^{\Lambda} \in SAT^{\Lambda}$.
Finally, we need to prove that for $\forall \overline{R} \in SN^{\Lambda}$, $\forall Q \in SN^{\Lambda}$ and $\forall K \in SN^{\Pi}$,

$$([Q/x]P)\overline{R} \in [\![(a_1, a_2)]\!]^{\Lambda} \Longrightarrow (\lambda x : K.P)Q\overline{R} \in [\![(a_1, a_2)]\!]^{\Lambda}$$

Since $([Q/x]P)\overline{R} \in [\![(a_1, a_2)]\!]^{\Lambda}$, we have $([Q/x]P)\overline{R}N \in [\![a_2]\!]^{\Lambda}$ for $\forall N \in [\![a_1]\!]^{\Lambda}$. And since $[\![a_1]\!]^{\Lambda} \subseteq SN^{\Lambda}$ and $[\![a_2]\!]^{\Lambda} \in SAT^{\Lambda}$, we have $(\lambda x : K.P)Q\overline{R}N \in [\![a_2]\!]^{\Lambda}$ and hence

$$(\lambda x : K.P)Q\overline{R} \in [\![(a_1, a_2)]\!]^{\Lambda}$$

- The proof of $[\![a]\!]^{\Pi} \in SAT^{\Pi}$ is similar to that of $[\![a]\!]^{\Lambda} \in SAT^{\Lambda}$ □

Notation: We often use SN for $SN^{\Lambda} \cup SN^{\Pi}$ and $[\![a]\!]$ for $[\![a]\!]^{\Lambda} \cup [\![a]\!]^{\Pi}$.

Definition 6. ***(Valuation)***

- *A* valuation *is a map $\rho : V \to \Lambda$, where V is the set of all term variables.*
- *Let ρ be a valuation. Then*

$$[\![M]\!]_{\rho} =_{df} [\rho(x_1)/x_1, ..., \rho(x_n)/x_n]M$$

where $x_1, ..., x_n$ are all of the free variable in M.
- *Let ρ be a valuation. Then*
  - · *ρ satisfies $M : a$, notation $\rho \models M : a$, if $[\![M]\!]_{\rho} \in [\![a]\!]$;*
  - · *ρ satisfies A, notation $\rho \models A$, if $\rho \models x : a$ for all $x : a \in A$;*
  - · *A satisfies $M : a$, notation $A \models M : a$, if*

$$\forall \rho \ (\rho \models A \Longrightarrow \rho \models M : a)$$

Remark 7. For any valuation ρ, if M is a term, $[\![M]\!]_{\rho}$ is also a term, and similarly, if M is a kind, $[\![M]\!]_{\rho}$ is also a kind. If a valuation ρ satisfies that $\rho(x) = x$ then $[\![M]\!]_{\rho} \equiv M$.

Lemma 3. ***(Soundness)*** *$A \vdash M : a \Longrightarrow A \models M : a$ where M is a term or kind.*

Proof. By induction on the derivations of $A \vdash M : a$.

1. The last rule is
$$\frac{A\ \mathit{valid}}{A \vdash \mathit{Type} : \mathit{Zero}}$$
Since $\llbracket \mathit{Type} \rrbracket_\rho = \mathit{Type}$ for any ρ and $\mathit{Type} \in SN = \llbracket \mathit{Zero} \rrbracket$, we have $\llbracket \mathit{Type} \rrbracket_\rho \in \llbracket \mathit{Zero} \rrbracket$.
2. The last rule is
$$\frac{A \vdash M : \mathit{Zero}}{A \vdash \mathit{El}(M) : \mathit{Zero}}$$
Since $\llbracket \mathit{El}(M) \rrbracket_\rho = \mathit{El}(\llbracket M \rrbracket_\rho)$ for any ρ and $\llbracket M \rrbracket_\rho \in \llbracket \mathit{Zero} \rrbracket = SN$, we have $\llbracket \mathit{El}(M) \rrbracket_\rho \in SN = \llbracket \mathit{Zero} \rrbracket$.
3. The last rule is
$$\frac{A \vdash K_1 : a_1 \quad A, x : a_1 \vdash K_2 : a_2}{A \vdash (x : K_1)K_2 : (a_1, a_2)}$$
We must show that
$$\forall \rho\ (\rho \models A \Longrightarrow \rho \models (x : K_1)K_2 : (a_1, a_2))$$
That is, we must show that $\llbracket (x : K_1)K_2 \rrbracket_\rho \in \llbracket (a_1, a_2) \rrbracket^{\Pi}$. By the definition of $\llbracket (a_1, a_2) \rrbracket^{\Pi}$, we must show that, for all $N \in \llbracket a_1 \rrbracket^{\Lambda}$,
$$\llbracket (x : K_1)K_2 \rrbracket_\rho N \in \llbracket a_2 \rrbracket^{\Pi}$$
Note that
$$\begin{array}{rcl} \llbracket (x : K_1)K_2 \rrbracket_\rho N & \equiv & ((x : K_1')K_2')N \\ & \to_{\beta_2} & [N/x]K_2' \\ & \equiv & \llbracket K_2 \rrbracket_{\rho \cup (N/x)} \end{array}$$
where $K_1' \equiv \llbracket K_1 \rrbracket_\rho \equiv [\rho(y_i)/y_i ...]K_1$ and $K_2' \equiv \llbracket K_2 \rrbracket_\rho \equiv [\rho(y_i)/y_i ...]K_2$ Now, let's consider the induction hypothesis. Since $\rho \cup (N/x) \models A, x : a_1$, we have $\llbracket K_1 \rrbracket_\rho \in \llbracket a_1 \rrbracket^{\Pi}$ and $\llbracket K_2 \rrbracket_{\rho \cup (N/x)} \in \llbracket a_2 \rrbracket^{\Pi}$. So, we have $[N/x]K_2' \in \llbracket a_2 \rrbracket^{\Pi}$, and because $\llbracket a_2 \rrbracket^{\Pi}$ is saturated, we have $((x : K_1')K_2')N \in \llbracket a_2 \rrbracket^{\Pi}$, i.e. $\llbracket (x : K_1)K_2 \rrbracket_\rho N \in \llbracket a_2 \rrbracket^{\Pi}$. Note that, since $\llbracket a_1 \rrbracket^{\Lambda} \subseteq SN^{\Lambda}$ and $\llbracket a_1 \rrbracket^{\Pi} \subseteq SN^{\Pi}$, we know that $N \in SN^{\Lambda}$ and $K_1' \in SN^{\Pi}$.
4. The last rule is
$$\frac{A \vdash K : (a_1, a_2) \quad A \vdash N : a_1}{A \vdash KN : a_2}$$

We must show that

$$\forall\rho\ (\rho \models A \Longrightarrow \rho \models KN : a_2)$$

By induction hypothesis, we have $[\![K]\!]_\rho \in [\![(a_1, a_2)]\!]^\Pi$ and $[\![N]\!]_\rho \in [\![a_1]\!]^\Lambda$.
By the definition of $[\![(a_1, a_2)]\!]^\Pi$, we have $[\![K]\!]_\rho [\![N]\!]_\rho \in [\![a_2]\!]^\Pi$, *i.e.* $[\![KN]\!]_\rho \in [\![a_2]\!]^\Pi$.

5. The last rule is

$$\frac{A, x : a, A'\ \mathit{valid}}{A, x : a, A' \vdash x : a}$$

Easy.

6. The last rule is

$$\frac{A \vdash K : a_1 \quad A, x : a_1 \vdash M : a_2}{A \vdash \lambda x : K.M : (a_1, a_2)}$$

Similar to case 3.

7. The last rule is

$$\frac{A \vdash M : (a_1, a_2) \quad A \vdash N : a_1}{A \vdash MN : a_2}$$

Similar to case 4. □

Theorem 1. *If $A \vdash M : a$, then M is strongly normalising.*

Proof. By Lemma 3 and take the evaluation ρ_0 that satisfies $\rho_0(x) = x$.
By Lemma 3, we have $A \models M : a$. So, by definition, we have

$$\rho_0 \models A \Longrightarrow \rho_0 \models M : a$$

Suppose $A \equiv x_1 : a_1, ..., x_n : a_n$. Since $[\![a_i]\!]^\Lambda \in SAT^\Lambda$, we have $x_i \in [\![a_i]\!]^\Lambda$. Hence $\rho_0 \models A$. So, we have $\rho_0 \models M : a$ and hence $M = [\![M]\!]_{\rho_0} \in [\![a]\!] \subseteq SN$. □

Translation from kinds to arities

Now, we define a map to translate kinds to arities, and prove that well-typed terms have correct arities.

Definition 7. *A map $arity : \Pi \to \Omega$ is inductively defined as follows.*

- $arity(Type) = Zero$,
- $arity(El(A)) = Zero$,
- $arity((x : K_1)K_2) = (arity(K_1), arity(K_2))$.

Notation: Suppose a context $\Gamma \equiv x_1 : K_1, ..., x_n : K_n$, then $arity(\Gamma) \equiv x_1 : arity(K_1), ..., x_n : arity(K_n)$.

Theorem 2. (Well-typed terms have correct arities) *If $\Gamma \vdash M : K$ is derivable in MLF, then $arity(\Gamma) \vdash M : arity(K)$ is derivable.*

Proof. By induction on the derivations of $\Gamma \vdash M : K$ (see the inference rules of MLF in Appendix).

Theorem 3. *If $\Gamma \vdash M : K$ is derivable in MLF, then M is strongly normalising.*

Proof. By Theorem 1 and Theorem 2.

5 Computation rules

In this section, we shall introduce computation rules for the type of dependent pairs and finite types and simple computation rules for universes. The strong normalisation is proved in a way that no one has ever take before in dependently typed systems, to the author's best knowledge. Recall that adding new computation (or reduction) rules will not increase the set of terms with correct arities. The basic strategy we adopt is to prove strong normalisation one reduction rule after another. That is, if we have already proved strong normalisation for a set of reduction rules, after adding one new reduction rule, can we still prove strong normalisation? This strategy will not work for dependently typed systems if we want to prove the statement that "well-typed terms are strongly normalising", because whenever we add a single computation rule, the set of well-typed terms may increase.

5.1 The type of dependent pairs

In MLF, the constants and computation rules for the type of dependent pairs can be specified as follows:

$$\begin{aligned}
\Sigma &: (A : Type)(B : (A)Type)Type \\
pair &: (A : Type)(B : (A)Type)(a : A)(b : B(a))\Sigma(A, B) \\
\pi_1 &: (A : Type)(B : (A)Type)(z : \Sigma(A, B))A \\
\pi_2 &: (A : Type)(B : (A)Type)(z : \Sigma(A, B))B(\pi_1(A, B, z))
\end{aligned}$$

$$\begin{aligned}
\pi_1(A, B, pair(A, B, a, b)) = a &\;:\; A \\
\pi_2(A, B, pair(A, B, a, b)) = b &\;:\; B(a)
\end{aligned}$$

In the arity system of the paper, we change the kinds to arities and the constants and the reduction rules are introduced as the following:

$$\begin{aligned}
\Sigma &: (Zero, ((Zero, Zero), Zero)) \\
pair &: (Zero, ((Zero, Zero), (Zero, (Zero, Zero)))) \\
\pi_1 &: (Zero, ((Zero, Zero), (Zero, Zero))) \\
\pi_2 &: (Zero, ((Zero, Zero), (Zero, Zero)))
\end{aligned}$$

$$\begin{aligned}
\pi_1(A, B, pair(A, B, a, b)) \longrightarrow_{\pi_1} a &\ :\ Zero \\
\pi_2(A, B, pair(A, B, a, b)) \longrightarrow_{\pi_2} b &\ :\ Zero
\end{aligned}$$

5.2 Finite types

In type systems, a finite type $\mathcal{T}$ can be represented by following constants

$$\begin{aligned}
\mathcal{T} &: Type \\
c_1 &: \mathcal{T} \\
&\dots \\
c_n &: \mathcal{T} \\
\mathcal{E}_{\mathcal{T}} &: (P : (\mathcal{T})Type) \\
&\quad (P(c_1))...(P(c_n)) \\
&\quad (z : \mathcal{T})(P(z))
\end{aligned}$$

and the following computation rules

$$\begin{aligned}
\mathcal{E}_{\mathcal{T}}(P, p_1, ..., p_n, c_1) = p_1 &\ :\ P(c_1) \\
&...... \\
\mathcal{E}_{\mathcal{T}}(P, p_1, ..., p_n, c_n) = p_n &\ :\ P(c_n)
\end{aligned}$$

In the arity system of the paper, we change the kinds to arities and the constants and the computation rules are introduced as follows.

$$\begin{aligned}
\mathcal{T} &: Zero \\
c_1 &: Zero \\
&\dots \\
c_n &: Zero \\
\mathcal{E}_{\mathcal{T}} &: ((Zero, Zero), \\
&\quad (Zero, (Zero, ...(Zero, \\
&\quad (Zero, Zero)...)
\end{aligned}$$

and the following reduction rules

$$\begin{aligned} \mathcal{E}_{\mathcal{T}}(P, p_1, ..., p_n, c_1) &\longrightarrow p_1 \;\; : \;\; Zero \\ &...... \\ \mathcal{E}_{\mathcal{T}}(P, p_1, ..., p_n, c_n) &\longrightarrow p_n \;\; : \;\; Zero \end{aligned}$$

Now, let's consider a concrete example, boolean type. Its representation in type systems and in the arity system are the following.

$$\begin{aligned} Bool &: Type \\ true &: Bool \\ false &: Bool \\ \mathcal{E}_{Bool} &: (P : (Bool)Type) \\ &\quad\;(p_1 : P(true))(p_2 : P(false)) \\ &\quad\;(z : Bool)P(z) \end{aligned}$$

$$\begin{aligned} \mathcal{E}_{Bool}(P, p_1, p_2, true) = p_1 \;\; &: \;\; P(true) \\ \mathcal{E}_{Bool}(P, p_1, p_2, false) = p_2 \;\; &: \;\; P(false) \end{aligned}$$

$$\begin{aligned} Bool &: Zero \\ true &: Zero \\ false &: Zero \\ \mathcal{E}_{Bool} &: ((Zero, Zero), \\ &\quad\;(Zero, (Zero, \\ &\quad\;(Zero, Zero)))) \end{aligned}$$

$$\begin{aligned} \mathcal{E}_{Bool}(P, p_1, p_2, true) \longrightarrow_{b_1} p_1 \;\; &: \;\; Zero \\ \mathcal{E}_{Bool}(P, p_1, p_2, false) \longrightarrow_{b_2} p_2 \;\; &: \;\; Zero \end{aligned}$$

5.3 Universe operator

We consider some simple case, for example,

$$\begin{aligned} U &: Type \\ Bool &: Type \\ bool &: U \\ uo &: (U)Type \end{aligned}$$

$$uo(bool) = Bool$$

$$\begin{aligned} U &: Zero \\ Bool &: Zero \\ bool &: Zero \\ uo &: (Zero, Zero) \end{aligned}$$

$$uo(bool) \longrightarrow_u Bool \;\; : \;\; Zero$$

5.4 Strong normalisation w.r.t. $\beta\eta\pi_1$-reduction

We have proved strong normalisation w.r.t. $\beta\eta$-reduction in Section 4. Now, we add the reduction rule π_1 and prove strong normalisation w.r.t. $\beta\eta\pi_1$-reduction. As mentioned before, the strategy is to prove strong normalisation one reduction rule after another. So after proving it w.r.t. $\beta\eta\pi_1$-reduction, we can add another rule (*eg,* π_2-reduction), and so on. In this section, we demonstrate the proof techniques through the proof w.r.t. $\beta\eta\pi_1$-reduction. For other reduction rules such as π_2, b_1, b_2 and u, the proof methods are the same.

Theorem 4. *If M doesn't have a correct arity under a context A without the π_1-reduction then M still doesn't have a correct arity under the context A with the π_1-reduction.*

Proof. The arities of the left hand side and the right hand side of the reduction rule π_1 are the same, and there is no reduction for arities. So, π_1-reduction becomes irrelevant whether M has a correct arity.

Remark 8. As mentioned before, in dependently typed systems, a term that is not well-typed can become a well-typed term after adding new reduction rules. For instance, under a context $f : (x : B(a))C$ and $y : B(\pi_1(pair(a, b)))$, the term $f(y)$ is not well-typed (some details are omitted here). However, if we add the π_1-reduction rule, then it becomes a well-typed term. This example shows that, after adding new reduction rules, well-typed terms may increase. This is one of the difficulties to prove the statement that "well-typed terms are strongly normalising".

Now, in order to prove strong normalisation, we prove some lemmas first.

Lemma 4. ***(Substitution for*** η***)*** *If $M_1 \longrightarrow_\eta M_2$ then $[N/x]M_1 \longrightarrow_\eta [N/x]M_2$. And if $N_1 \longrightarrow_\eta N_2$ then $[N_1/x]M \twoheadrightarrow_\eta [N_2/x]M$.*

Proof. For the first part, we proceed the proof by induction on M_1, and for the second part, by induction on M. In the case that M is a variable, we consider two sub-cases: $M \equiv x$ and $M \not\equiv x$.

Lemma 5. *If $M_1 \longrightarrow_\beta M_2$ and $x \notin FV(M_1)$ then $x \notin FV(M_2)$.*

Proof. By induction on M_1.

Lemma 6. *If $M_1 \longrightarrow_\eta \lambda x : K_2.M_2$ then there are three and only three possibilities as the following:*

- *$M_1 \equiv \lambda y : K_1.(\lambda x : K_2.M_2)y$ for some y and K_1, and $y \notin FV(\lambda x : K_2.M_2)$.*
- *$M_1 \equiv \lambda x : K_2.N$ for some N and $N \longrightarrow_\eta M_2$.*
- *$M_1 \equiv \lambda x : K_1.M_2$ for some K_1 and $K_1 \longrightarrow_\eta K_2$.*

Proof. By the understanding of one-step reduction.

Lemma 7. ***(Commutation for $\eta\beta$)*** *If $M_1 \longrightarrow_\eta M_2$ and $M_2 \longrightarrow_\beta M_3$ then there exists a M_2' such that $M_1 \twoheadrightarrow_\beta^+ M_2'$ and $M_2' \twoheadrightarrow_\eta M_3$.*

Proof. By induction on M_1 and Lemma 4, 5 and 6.

Lemma 8. ***(Substitution for π_1)*** *If $M_1 \longrightarrow_{\pi_1} M_2$ then $[N/x]M_1 \longrightarrow_{\pi_1} [N/x]M_2$. And if $N_1 \longrightarrow_{\pi_1} N_2$ then $[N_1/x]M \twoheadrightarrow_{\pi_1} [N_2/x]M$.*

Proof. Similar to the proof of Lemma 4.

Lemma 9. *If $M_1 \longrightarrow_{\pi_1} \lambda x : K_2.M_2$ then there are two and only two possibilities as the following:*

- *$M_1 \equiv \lambda x : K_2.N$ for some N and $N \longrightarrow_{\pi_1} M_2$.*
- *$M_1 \equiv \lambda x : K_1.M_2$ for some K_1 and $K_1 \longrightarrow_{\pi_1} K_2$.*

Proof. By the understanding of one-step reduction and the arity of M_1 is not *Zero*.

Lemma 10. ***(Commutation for $\pi_1\beta$)*** *If $M_1 \longrightarrow_{\pi_1} M_2$ and $M_2 \longrightarrow_\beta M_3$ then there exists a M_2' such that $M_1 \longrightarrow_\beta M_2'$ and $M_2' \twoheadrightarrow_{\pi_1} M_3$.*

Proof. By induction on M_1 and Lemma 8 and 9.

Theorem 5. *If $A \vdash M : a$, then M is strongly normalising w.r.t. $\beta\eta\pi_1$-reduction.*

Proof. We proceed the proof by contradiction, and by Theorem 1 and Lemma 7 and 10.

Suppose there is an infinite reduction sequence for M and it is called S. By Theorem 1, M is strongly normalising w.r.t. $\beta\eta$-reduction. So, S must contain infinite times of π_1-reduction. Every time when η-reduction or π_1-reduction rule is applied, terms become smaller. So, M is strongly normalising w.r.t. $\eta\pi_1$-reduction. And hence S must also contain infinite times of β-reduction. In fact, S must be like the following,

$$M \twoheadrightarrow^{+}_{\eta\pi_1} M_1 \twoheadrightarrow^{+}_{\beta} M_2 \twoheadrightarrow^{+}_{\eta\pi_1} M_3 \twoheadrightarrow^{+}_{\beta} M_4 \twoheadrightarrow^{+}_{\eta\pi_1} \ldots$$

or

$$M \twoheadrightarrow^{+}_{\beta} M_1 \twoheadrightarrow^{+}_{\eta\pi_1} M_2 \twoheadrightarrow^{+}_{\beta} M_3 \twoheadrightarrow^{+}_{\eta\pi_1} M_4 \twoheadrightarrow^{+}_{\beta} \ldots$$

where $\twoheadrightarrow^{+}_{\beta}$ means one or more but finite reduction steps of β, and similarly, $\twoheadrightarrow^{+}_{\eta\pi_1}$ means one or more but finite reduction steps of η or π_1.

Now, by Lemma 7 and Lemma 10, for the infinite sequence S, we can always move the β-reduction steps forward and build an infinite sequence of β-reduction. This is a contradiction to that M is strongly normalising w.r.t. β-reduction. □

6 Conclusions and future work

Strong normalisation for MLF has been proved in the paper, but we did not follow the traditional understanding, that is, well-typed terms are strongly normalising. Instead, a weaker condition has been proposed, which says terms with correct arities are strongly normalising. The author hopes this new understanding will inspire us to think the question "why is a term strongly normalising?" again, and to simplify the proofs for dependently typed systems.

Another important technique employed in the paper is that, in order to prove what we want, we prove a more general and stronger property. In the paper, the definition of terms and kinds is extended and a new reduction rule β_2 is introduced. And we proved strong normalisation w.r.t. $\beta\eta\beta_2$-reduction instead of w.r.t. $\beta\eta$-reduction only. This generalisation is quite different from the traditional idea of generalising induction hypothesis.

We only studied the computation rules for some inductive data types and these rules have commutation property. However, some computation rules do not have such property, for instance, the computation rule for the type of function space. How to prove strong normalisation for such rules needs further study. The question of how to develop weaker conditions to simplify the normalisation proofs for other type systems is also worth being taken into our consideration.

Acknowledgements Thanks to Zhaohui Luo, Sergei Soloviev, James McKinna and Healfdene Goguen for discussions on the issue of strong normalisation, and for reading the earlier version of the paper, and for their helpful comments and suggestions.

References

[Alt93] Th. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, Edinburgh University, 1993.

[Alt94] Thorsten Altenkirch. Proving strong normalization of CC by modifying realizability semantics. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs*, LNCS 806, pages 3 – 18, 1994.

[Bar84] H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics.* North-Holland, revised edition, 1984.

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Clarendon Press, 1992.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland Publishing Company, 1958.

[CG90] Th. Coquand and J.H. Gallier. A proof of strong normalization for the theory of constructions using a Kripke-like interpretation. In *Preliminary Proc. of the Workshop on Logical Frameworks*, Antibes, 1990.

[Coq85] Th. Coquand. *Une Theorie des Constructions.* PhD thesis, University of Paris VII, 1985.

[Cos96] R. Di Cosmo. On the power of simple diagrams. In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103, pages 200–214. Lecture Notes in Computer Science, 1996.

[dB80] N. G. de Bruijn. A survey of the project AUTOMATH. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[Geu93] Herman Geuvers. *Logics and Type Systems.* PhD thesis, Katholieke Universiteit Nijmegen, 1993.

[Gog94] H. Goguen. *A Typed Operational Semantics for Type Theory.* PhD thesis, University of Edinburgh, 1994.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Proc. 2nd Ann. Symp. on Logic in Computer Science. IEEE*, 1987.

[HHP92] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of ACM*, 40(1):143–184, 1992.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic.* Academic Press, 1980.

[Luo90] Z. Luo. *An Extended Calculus of Constructions.* PhD thesis, University of Edinburgh, 1990. Also as Report CST-65-90/ECS-LFCS-90-118, Department of Computer Science, University of Edinburgh.

[Luo94] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, 1994.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[MW96] P. Mellies and B. Werner. A generic normalisation proof for pure type systems, 1996.

[NPS90] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[SC04] S. Soloviev and D. Chemouil. Some algebraic structures in lambda-calculus with inductive types. In *Types for Proofs and Programs*, volume 3085. Lecture Notes in Computer Science, 2004.

[Wer92] B. Werner. A normalization proof for an impredicative type system with large eliminations over integers. In *Workshop on Logical Frameworks*, 1992.

Appendix

Terms and Kinds in MLF

- Terms
  1. a variable is a term,
  2. $\lambda x : K.M$ is a term if x is a variable, K is a kind and M is a term,
  3. MN is a term if M and N are terms.
- Kinds
  1. $Type$ is a kind,
  2. $El(M)$ is a kind if M is a term,
  3. $(x : K_1)K_2$ is a kind if K_1 and K_2 are kinds.

Reduction rules in MLF

$$(\lambda x : K.M)N \longrightarrow_\beta [N/x]M$$

$$\lambda x : K.Mx \longrightarrow_\eta M \qquad x \notin FV(M)$$

Inference rules for MLF

Contexts and assumptions

$$\frac{}{<>\ \ valid} \quad \frac{\Gamma \vdash K\ \ kind \quad x \notin FV(\Gamma)}{\Gamma, x : K\ \ valid} \quad \frac{\Gamma, x : K, \Gamma'\ \ valid}{\Gamma, x : K, \Gamma' \vdash x : K}$$

Equality rules

$$\frac{\Gamma \vdash K\ \ kind}{\Gamma \vdash K = K} \quad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \quad \frac{\Gamma \vdash K = K' \ \ \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$\frac{\Gamma \vdash k : K}{\Gamma \vdash k = k : K} \quad \frac{\Gamma \vdash k = k' : K}{\Gamma \vdash k' = k : K} \quad \frac{\Gamma \vdash k = k' : K \ \ \Gamma \vdash k' = k'' : K}{\Gamma \vdash k = k'' : K}$$

$$\frac{\Gamma \vdash k : K \ \ \Gamma \vdash K = K'}{\Gamma \vdash k : K'} \quad \frac{\Gamma \vdash k = k' : K \ \ \Gamma \vdash K = K'}{\Gamma \vdash k = k' : K'}$$

Substitution rules

$$\frac{\Gamma, x : K, \Gamma' \ \ valid \ \ \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \ \ valid}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash K' \ \ kind \ \ \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' \ \ kind} \quad \frac{\Gamma, x : K, \Gamma \vdash K' \ \ kind \ \ \Gamma \vdash k = k' : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k'/x]K'}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash k' : K' \ \ \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' : [k/x]K'} \quad \frac{\Gamma, x : K, \Gamma' \vdash k' : K' \ \ \Gamma \vdash k_1 = k_2 : K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]k' = [k_2/x]k' : [k_1/x]K'}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash K' = K'' \ \ \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k/x]K''} \quad \frac{\Gamma, x : K, \Gamma' \vdash k' = k'' : K' \ \ \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' = [k/x]k'' : [k/x]K'}$$

The kind type

$$\frac{\Gamma \ \ valid}{\Gamma \vdash Type \ \ kind} \quad \frac{\Gamma \vdash A : Type}{\Gamma \vdash El(A) \ \ kind} \quad \frac{\Gamma \vdash A = B : Type}{\Gamma \vdash El(A) = El(B)}$$

Dependent product kinds

$$\frac{\Gamma \vdash K \ \ kind \ \ \Gamma, x : K \vdash K' \ \ kind}{\Gamma \vdash (x : K)K' \ \ kind} \quad \frac{\Gamma \vdash K_1 = K_2 \ \ \Gamma, x : K_1 \vdash K'_1 = K'_2}{\Gamma \vdash (x : K_1)K'_1 = (x : K_2)K'_2}$$

$$\frac{\Gamma, x : K \vdash k : K'}{\Gamma \vdash \lambda x : K.k : (x : K)K'} \quad (\xi)\, \frac{\Gamma \vdash K_1 = K_2 \ \ \Gamma, x : K_1 \vdash k_1 = k_2 : K}{\Gamma \vdash \lambda x : K_1.k_1 = \lambda x : K_2.k_2 : (x : K_1)K}$$

$$\frac{\Gamma \vdash f : (x : K)K' \ \ \Gamma \vdash k : K}{\Gamma \vdash f(k) : [k/x]K'} \quad \frac{\Gamma \vdash f = f' : (x : K)K' \ \ \Gamma \vdash k_1 = k_2 : K}{\Gamma \vdash f(k_1) = f'(k_2) : [k_1/x]K'}$$

$$(\beta)\frac{\Gamma, x : K \vdash k' : K' \ \ \Gamma \vdash k : K}{\Gamma \vdash (\lambda x : K.k')(k) = [k/x]k' : [k/x]K'} (\eta)\frac{\Gamma \vdash f : (x : K)K' \ \ x \notin FV(f)}{\Gamma \vdash \lambda x : K.f(x) = f : (x : K)K'}$$