A NOTE ON A STRAIGHTFORWARD PROOF OF NORMAL FORM THEOREM FOR SIMPLY TYPED λ-CALCULI

SILVIO VALENTINI

Dipartimento di Matematica Pura ed Applicata
Università di Padova

SUNTO. Il lavoro presenta una nuova prova, basata su alcune idea di Martin Löf e Tait, del ben noto teorema di forma normale per il λ-calcolo tipato semplice, la cui prima dimostrazione è dovuta a Turing. Il metodo di prova usato è molto semplice, ma al tempo stesso si presenta come molto generale e facilmente applicabile ad una varietà di λ-calcoli tipati.

Key words and phrases. Typed λ-calculus, Normal-form theorem.

Summary

We will illustrate a method to obtain the normal-form theorem for simply typed λ-calculi (see [Bar]) which is both straightforward and general.

1. Introduction

In this paper we will describe a new method to prove the standard result of normalizability for simply typed λ-calculi due to Turing. Even if the method seems to be of general applicability, in order to make the exposition more easy to understand we will illustrate the very elementary example where only function types can be constructed over some basic types[1].

Definition: (Types).

(Basic types) $$\frac{C \text{ basic type}}{C \text{ type}}$$

(Arrow types) $$\frac{\alpha \text{ type} \qquad \beta \text{ type}}{\alpha \to \beta \text{ type}}$$

In this case one can assume to have variables of type α (notation $x : \alpha$ var), for any type α, and one can form the following lambda expressions, in a context which comprises all the variables used in the lambda expression[2] , and, at the same time, collect the variables which appear free within them.

Definition: (λ-expression).

(var-rule)
$$x : \alpha \ \lambda\text{-exp } [\Gamma, x : \alpha \text{ var}] \qquad FV(x) = \{x\}$$
(app-rule)
$$\frac{b : \alpha \to \beta \ \lambda\text{-exp } [\Gamma] \quad a : \alpha \ \lambda\text{-exp } [\Gamma]}{b(a) : \beta \ \lambda\text{-exp } [\Gamma]} \qquad FV(b(a)) = FV(b) \cup FV(a)$$
(abst-rule)
$$\frac{b : \beta \ \lambda\text{-exp } [\Gamma, x : \alpha \text{ var}]}{(\lambda x.b) : \alpha \to \beta \ \lambda\text{-exp } [\Gamma]} \qquad FV((\lambda x.b)) = FV(b) \setminus \{x\}$$

The substitution of a variable by a term within a lambda expression is introduced in the usual way.

Definition: (Substitution). Let $x : \alpha$ var, $e : \alpha$ λ-exp and $b : \beta$ λ-exp, then the substitution of the variable x by the expression e in the expression b (notation $b[x := e]$)

[1] A more general approach can be found in [Bos-Val].

[2] We assume the reader is aware of the standard operations of thining, contraction and exchange between the assumptions in a context.

is defined by induction on the structural complexity of b:

(variable step) $$z[x := e] \equiv \begin{cases} e & \text{if } z \equiv x \\ z & \text{otherwise} \end{cases}$$

(application step) $$b(a)[x := e] \equiv b[x := e](a[x := e])$$

(abstraction step)

$$(\lambda z.b)[x := e] \equiv \begin{cases} (\lambda x.b) & \text{if } z \equiv x \\ (\lambda z.b[x := e]) & \text{if } z \not\equiv x \text{ and } z \notin FV(e) \\ (\lambda t.b[z := t][x := e]) & \text{otherwise} \\ \text{where } t \text{ is a new variable of the same type of } z \end{cases}$$

Finally we recall that the usual equality between two λ-expressions induced by $\beta\eta$-conversion[3] is the minimal equivalence relation such that the following conditions hold.

Definition: (λ-equality).

(var-equality)

$$x = x : \alpha\ [\Gamma, x : \alpha \text{ var}]$$

(app-equality)

$$\frac{b_1 = b_2 : \alpha \to \beta\ [\Gamma] \quad a_1 = a_2 : \alpha\ [\Gamma]}{b_1(a_1) = b_2(a_2) : \beta\ [\Gamma]}$$

(β-equality)

$$\frac{b : \beta\ \lambda\text{-exp}\ [\Gamma, x : \alpha \text{ var}] \quad a : \alpha\ \lambda\text{-exp}\ [\Gamma]}{(\lambda x.b)(a) = b[x := a] : \beta\ [\Gamma]}$$

(ξ-equality)

$$\frac{b = d : \beta\ [\Gamma, x : \alpha \text{ var}]}{(\lambda x.b) = (\lambda x.d) : \alpha \to \beta\ [\Gamma]}$$

(η-equality)

$$\frac{b : \alpha \to \beta\ \lambda\text{-exp}\ [\Gamma] \quad x : \alpha \text{ var}}{(\lambda x.b(x)) = b : \alpha \to \beta\ [\Gamma]} \quad \text{provided } x \notin FV(b)$$

2. Normal form.

The notion of normal form is central in the development of a simply typed λ-calculus: it establishes a canonical form for the expressions, i.e. their *simplest* form. In the case we are considering only one form of simplification is introduced.

Definition: (β-reduction). $(\lambda x.b)(a) \Rightarrow b[x := a]$

The simplest form of an expression is obviously the one where no reduction can be applied.

[3] As usual, we assume that two expressions which differ only for the name of the abstracted variables are identical, i.e. the expressions $(\lambda x.b)$ and $(\lambda y.b[x := y])$, where $y \notin FV(b)$, coincide.

Definition: (Normal form). An expression is in normal form if no β-reduction can be applied to any of its sub-expressions.

The following algorithm, provided it is totally correct, shows a method to obtain, for any given expression, an equivalent expression in normal form.

Algorithm: (Conversion into normal form). Let e be an expression of type α and consider the following recursive definition.

$$nf(e) = \begin{cases} (\lambda x.nf(e(x))) & \text{if } \alpha \equiv \beta \to \gamma \text{ where } x \text{ is a new variable of type } \beta \\ x(nf(b_1))\dots(nf(b_n)) & \text{if } \alpha \equiv C \text{ and } e \equiv x(b_1)\dots(b_n) \\ nf(c[x := a](b_1)\dots(b_n)) & \text{if } \alpha \equiv C \text{ and } e \equiv (\lambda x.c)(a)(b_1)\dots(b_n) \end{cases}$$

The proof of partial correctness is easy.

Theorem: (Partial correctness). *Let e be an expression of type α, then $nf(e)$ is an expression of type α in normal form equivalent to e.*

Proof. Provided the algorithm of conversion into normal form terminates the result is almost obvious. In fact obviously $nf(e)$ is an expression of type α such that no β-reduction can be applied to any of its sub-expressions. Moreover induction on the number of steps of the algorithm can be used to prove equivalence between e and $nf(e)$.

3. A new system to derive expressions

Then the problem is to prove termination of the algorithm of conversion into normal form. To this aim we will define a new system to derive expressions such that to form an expression one need to have formed exactly the expressions that are needed to prove its normalizability.

Definition: (Expression).

$$(1) \qquad \frac{a_1 : \alpha_1 \text{ exp } [\Gamma] \dots a_n : \alpha_n \text{ exp } [\Gamma]}{x(a_1)\dots(a_n) : C \text{ exp } [\Gamma, x : \alpha_1 \to (\dots(\alpha_n \to C)\dots) \text{ var}]} \qquad n \geq 0$$

$$(2) \qquad \frac{c\ [x := a](b_1)\dots(b_n) : C\ [\Gamma] \quad a : \alpha \text{ exp } [\Gamma]}{(\lambda x.c)(a)(b_1)\dots(b_n) : C \text{ exp } [\Gamma]} \qquad n \geq 0$$

$$(3) \qquad \frac{b(x) : \beta \text{ exp } [\Gamma, x : \alpha \text{ var}]}{b : \alpha \to \beta \text{ exp } [\Gamma]}$$

where in 3. the only occurrence of x in $b(x)$ is the manifested one.

It is obvious that the algorithm of normalization of the previous section terminates if applied to one of this expression (just use induction on its derivation). Hence to conclude the proof of normalizability of each λ-expression we must only show that each λ-expression is an expression.

First note that it is easy to prove by induction on the type complexity the following lemma that shows closure of the expression system under var-rule.

Lemma: (Closure under var-rule). *If $x : \alpha$ var then $x : \alpha$ exp.*

The next step is to prove closure of the expression system under substitution.

Theorem: (Closure under substitution). *Let $b : \beta$ be an expression, $x : \alpha$ a variable and $a : \alpha$ an expression, then $b[x := a] : \beta$ is an expression.*

Proof. By principal induction on the complexity of the type α and secondary induction on the length of the derivation of $b : \beta$. The proof is straightforward if b is obtained by 3. while to deal with case 1. the principal induction is needed. Finally to deal with case 2. one must observe that, for any expression c, $c[x := d][x := a] \equiv c[x := d[x := a]]$, $c[y := d][x := a] \equiv c[x := a][y := d[x := a]]$, provided y does not appear free in a, and $c[y := d][x := a] \equiv c[y := t][x := a][t := d[x := a]]$, provided t does not appear free in a.

Now the missing link can easily be established.

Theorem. *Let $b : \beta$ λ-exp then $b : \beta$ exp.*

Proof. By induction on the length of the derivation of $b : \beta$ λ-exp. We already proved closure under var-rule and, using the theorem on closure under substitution, it is easy to prove closure under app-rule since if $b : \alpha \to \beta$ is an expression then it must be formed from $b(x) : \beta$ for a suitable choice of the variable x of type α. Finally suppose $b : \beta$ is an expression, then it must be formed from $b(x_1) \ldots (x_n) : C$, for a suitable choice of the variables $x_1, \ldots, x_n$. Since $b \equiv b[x := x]$, by using 2. we obtain $(\lambda x.b)(x)(x_1) \ldots (x_n) : C$ and hence closure under abst-rule follows by repeated applications of 3.

Conclusion

The method we described is very simple and just the mandatory steps in a normalization proof are involved. Moreover it is easy to enhance it in order to consider also more complex kind of simply typed λ-calculi. Let us, for instance, consider the typed λ-calculus obtained from the previous one by adding also cartesian product:

(product types) $$\frac{\alpha_1 \text{ type } \ldots \ \alpha_n \text{ type}}{\alpha_1 \times \cdots \times \alpha_n \text{ type}}$$

and hence the term formation rules:

(prod-rule) $$\frac{a_1 : \alpha_1 \ \lambda\text{-exp } [\Gamma] \ \ldots \ a_n : \alpha_n \ \lambda\text{-exp } [\Gamma]}{\langle a_1, \ldots, a_n \rangle : \alpha_1 \times \cdots \times \alpha_n \ \lambda\text{-exp } [\Gamma]}$$

(sel-rule) $$\frac{a : \alpha_1 \times \cdots \times \alpha_n \ \lambda\text{-exp } [\Gamma]}{\{a\}_i : \alpha_i \ \lambda\text{-exp } [\Gamma]} \quad 1 \leq i \leq n$$

and the reduction rule

(selection) $$\{\langle a_1, \ldots, a_n \rangle\}_i \Rightarrow a_i \qquad 1 \leq i \leq n$$

Then, it is easy to modify our rules to form expressions in order to be able to prove total correctness of the obvious normalization algorithm, by adding the rules

(4) $$\frac{a_i \cdots : C \text{ exp } [\Gamma] \qquad a_1 : \alpha_1 \text{ exp } [\Gamma] \ \ldots \ a_n : \alpha_n \text{ exp } [\Gamma]}{\{\langle a_1, \ldots, a_n \rangle\}_i \cdots : C \text{ exp } [\Gamma]}$$

(5) $$\frac{\{a\}_1 : \alpha_1 \text{ exp } [\Gamma] \ \ldots \ \{a\}_n : \alpha_n \text{ exp } [\Gamma]}{a : \alpha_1 \times \cdots \times \alpha_n \text{ exp } [\Gamma]}$$

Two interesting problems are not considered in this paper: can this normalization proof be easily transformed into a strong normalization proof? Is it possible to work out a similar proof for second order typed λ-calculus in the style of Girard's *System F* [Gir-Laf-Tay]?

References

[Bar] H.P.Barendregt, *The Lambda Calculus : its syntax and semantics*, North Holland, Amsterdam, 1981.

[Bos-Val] A.Bossi, S.Valentini, *The expressions with arity*, rapporto interno **61/89** (1989), Dip. Scienze dell'Informazione, Univ. di Milano, Milano.

[Gir-Laf-Tay] J.Y.Girard, Y.Lafont, P.Taylor, *Proofs and Types*, Cambridge University Press, Cambridge, 1989.

Dipartimento di Matematica Pura ed Applicata, Università di Padova, via G. Belzoni n.7, 35131 Padova (ITALY)
E-mail address: valentini@pdmat1.unipd.it