A proof of the normal form theorem for the closed terms of Girard's System F by means of computability

Silvio Valentini
Dip. Matematica Pure ed Applicata
Università di Padova
via G.Belzoni 7, I-35131 Padova (ITALY)
email valentini@pdmat1.unipd.it

Summary

In this paper a proof of the normal form theorem for the closed terms of Girard's system F is given by using a computability method 'a la' Tait. It is worth noting that most of the standard consequences of the normal form theorem can be obtained using this version of the theorem as well. From the proof-theoretical point of view the interest of the proof is that the definition of computable derivation here used does not seem to be well founded.

1. Introduction.

Let us recall the main facts about system F (see [Gir-Laf-Tay]) using an expository style which will make the definitions later in the text easier.

We deal with two forms of judgement[1]:

- α type [Γ], i.e. "α is a type in the context Γ".
- a:α [Γ], i.e. "a is a term of type α in the context Γ".

In both the forms of judgement the context is an ordered set of assumptions of individual and type variables which includes all the variables which appear free in the type α or in the term a. The order among assumptions requires that all the type variables which appear free in the type of the assumption of an individual variable are already assumed (i.e. they appear at its left in the context). Note that no individual variable can appear in a type since system F does not allow dependent types. We assume the standard rules of weakening, contraction and exchange among the assumptions of a context as long as the order among assumptions in the context is fulfilled.

Girard supposes to have a numerable set of variables for types (here the notation X typeVar is used) and to construct the types by the following rules.

[1] We follow here the terminology used in [MLöf] and [Bo-Val]

Type formation

(variable type) $\quad X \text{ type } [X \text{ typeVar}]$

(arrow type) $\quad \dfrac{\alpha \text{ type } [\Gamma] \quad \beta \text{ type } [\Gamma]}{\alpha \to \beta \text{ type } [\Gamma]}$

(quantification) $\quad \dfrac{\beta \text{ type } [\Gamma, X \text{ typeVar}]}{\Pi X.\beta \text{ type } [\Gamma]}$

where quantification is the only rule which bounds a type variable.

Then the terms are introduced.

Term formation

var) $\dfrac{\alpha \text{ type } [\Gamma]}{x{:}\alpha \ [\Gamma, x{:}\alpha]}$

abst1) $\dfrac{b{:}\beta \ [\Gamma, x{:}\alpha]}{(\lambda x{:}\alpha.b){:}\alpha \to \beta \ [\Gamma]}$ app1) $\dfrac{c{:}\alpha \to \beta \ [\Gamma] \quad a{:}\alpha \ [\Gamma]}{\mathrm{ap}(c,a){:}\beta \ [\Gamma]}$

abst2) $\dfrac{b{:}\beta \ [\Gamma, X \text{ typeVar}]}{(\Lambda X.b){:}\Pi X.\beta \ [\Gamma]}$ app2) $\dfrac{c{:}\Pi X.\beta \ [\Gamma] \quad \alpha \text{ type } [\Gamma]}{\mathrm{AP}(c,\alpha){:}\beta[X{:=}\alpha] \ [\Gamma]}$

where in abst2, following the convention on the order among assumptions of a context, we mean that in Γ no element variable is assumed to belong to a type where X appears free.

In order to complete the exposition of system F we have to specify the rules which allow to simplify a term into an equivalent one in normal form. Here we do not follow the standard approach but we prefer to give weaker rules than the usual β reductions. Adopting a notation used by Martin Löf (see [Bo-Val]), they are:

$(\lambda x{:}\alpha.\ b) \quad (\lambda x{:}\alpha.\ b)$

$(\Lambda X.\ b) \quad (\Lambda X.\ b)$

$$\frac{c \quad (\lambda x{:}\alpha.\ b) \quad b[x{:=}a] \quad g}{\mathrm{ap}(c,a) \quad g}$$

$$\frac{c \quad (\Lambda X.\ b) \quad b[X{:=}\alpha] \quad g}{\mathrm{AP}(c,\alpha) \quad g}$$

It is easy to realize that these rules are the usual β-reduction ones with the restriction that a rule is always “lazy”, that is the reduction process stops as soon as the external form of a term is normal. Moreover the rules do not handle variables, i.e. they can only be applied to closed terms.

Obviously the application of the rules to a term *t* gives rise to a tree. If this tree is finite, i.e. the evaluation process terminates, the term *t* will be said to have a value.

Our version of the normal form theorem can now be stated.

Theorem: (Normal form theorem)

Any closed term which is proved to belong to a type has a value.

Girard's exposition of system F is very clear and elegant; moreover it is well known that the system is very powerful since it can represent a wide variety of data structures and recursive functions. But it requires a very involved proof of the normal form theorem: due to Gödel's theorem this is not unexpected since the normal form theorem for system F implies the consistency of second order arithmetic.

2. Computability

In order to prove the normal form theorem let us begin by giving the definition of suitable substitution.

Definition 2.1: (Suitable substitution)

Let Γ be a context; then G is a *suitable substitution for* Γ if any judgement in G is derivable and

Γ ≡ X typeVar	and	G ≡ α type
Γ ≡ *x*:α	and	G ≡ *a*:α
Γ ≡ X typeVar, Γ'	and	G ≡ α type, G' where G' is a suitable substitution for Γ'[X:=α]
Γ ≡ *x*:α, Γ'	and	G ≡ *a*:α, G' where G' is a suitable substitution for Γ'[*x*:=*a*].

Note that in the last clause of this definition Γ'[*x*:=*a*] ≡ Γ' since system F does not allow dependent types.

An example of suitable substitution for the context "X typeVar, *x*:X, *y*:X→X" is "α type, *a*:α, *b*:α→α", provided that these judgements are derivable.

In order to denote the result of applying the suitable substitution G for the context Γ to the judgement *a*:α[Γ] (or α type [Γ]) we will write *a*[Γ:=G]:α[Γ:=G] (respectively α[Γ:=G] type). Although the definition of suitable substitution is not the most general notion of substitution one can consider, it is sufficient for us; in fact we are interested in substituting the free variables in a judgement by using only *closed judgements*, i.e. judgements with an empty context.

Lemma 2.2: (Head substitution)

Let J[Γ, Δ] be a derivable judgement and G be a suitable substitution for Γ; then the judgement J[Γ:=G] [Δ[Γ:=G]] is derivable.

Proof: By induction on the length of the derivation of J[Γ,Δ]. Almost all cases are straightforward and for this reason only the most interesting are shown here.

- (quantification) If β type [Γ, Δ, X typeVar] is derivable then, by inductive hypothesis, β[Γ:=G] type [Δ[Γ:=G], X typeVar] is derivable (note that X can not appear in the context Γ) hence ΠX.(β[Γ:=G])≡(ΠX.β)[Γ:=G] type [Δ[Γ:=G]] is derivable.
- (assumption) If α type [Γ, Δ] is derivable then two cases must be considered:

(1) G is a suitable substitution for Γ, hence,by inductive hypothesis, α[Γ:=G] type [Δ[Γ:=G]] is derivable and thus also *x*:α[Γ:=G] [Δ[Γ:=G], *x*:α[Γ:=G]] is derivable;

(2) $G \equiv G'$, $a{:}\alpha[\Gamma':=G']$ is a suitable substitution for $\Gamma \equiv \Gamma'$, $x{:}\alpha$ then, because of the order among assumption condition, α depends by no assumption in Δ, hence $a{:}\alpha[\Gamma:=G]$ is derivable and thus also $a{:}\alpha[\Gamma:=G]\ [\Delta[\Gamma:=G]]$ is derivable.

At last we have arrived at the main definition of the paper.

Definition 2.3: (Computable judgement)
Let $a{:}\ \alpha\ [\Gamma]$ be a derivable judgement; then it is computable if
Case $\Gamma=\emptyset$

$\alpha\equiv\gamma\rightarrow\beta$ if $a\ \ (\lambda x{:}\gamma.\ b)$
$b{:}\beta\ [x{:}\gamma]$ is computable

$\alpha\equiv\Pi X.\beta$ if $a\ \ (\Lambda X.\ b)$
$b{:}\beta$ [X typeVar] is computable

Case $\Gamma\neq\emptyset$

the judgement $a[\Gamma:=G]{:}\ \alpha[\Gamma:=G]$ is computable for any suitable substitution G for Γ such that the assumptions of the element variables are substituted with computable judgements (we will say that G is a *computable suitable substitution* for Γ, in short c.s.s.).

First of all it must be pointed out that this definition does not seem to be predicative, differently from the usual approach to the definition of computable judgement (see for instance [Bo-Val]) where the analogous definition is given by double induction on the complexity of the type and the number of assumptions in the judgement. For instance the judgement $a{:}\Pi X.X$ is computable if and only if $a\ \ (\Lambda X.b)$ and $b{:}X$ [X typeVar] is computable and this holds if and only if $b[X:=\beta]{:}\beta$ is computable for any type β: it is very hard to affirm that the notion of computability of the judgement in the last statement can be easier understood than the original one since the type β can be much more complex than the type $\Pi X.X$ we started with. Of course, the real problem is that no order among type complexities can be consistently defined for the types of system F. On the other hand, in a standard proof meant to verify that any derivable judgement is computable, an induction on the computational complexity[2] is usually required but this is not the case for system F.

Our purpose is now to demonstrate that every derivable judgement is computable. In order to obtain this result one has to observe that even if the inductive definition of computable judgement does not seem to consider any basic step, this is not the case. In fact, the second clause in the definition shows that any non-closed judgement is computable if no c.s.s. can be found for its context. Moreover we also have the following lemma which shows that any assumption is computable.

[2] Some kind of measure of the complexity involved in deciding whether a judgement is computable.

Lemma 2.4:

If α type [Γ] is derivable then x:α [Γ, x:α] is computable.

Proof: Let "G, a:α[Γ:=G]" be a c.s.s. for "Γ, x:α"; then, in particular, a:α[Γ:=G] is computable but this is just the judgement which results from the substitution.

Now we will prove that each rule preserves computability, i.e. if the judgements in its premises are computable also the one in the conclusion is. To this aim, we need to prove first that the head substitutions preserve computability.

Lemma 2.5:

Let b:β [Γ, X typeVar] be a computable judgement and G be a c.s.s. for Γ; then the judgement b[Γ:=G]: β[Γ:=G] [X typeVar] is computable.

Proof: Let α type be a c.s.s. for X typeVar; then "G, α type" is a c.s.s. for "Γ, X typeVar" hence b[Γ,X := G,α type] ≡ b[Γ:=G][X:=α type]:β[Γ,X := G,α type] ≡ β[Γ:=G][X :=α type] is computable.

Lemma 2.6:

Let b: β [Γ, x:α] be a computable judgement and G be a c.s.s. for Γ; then the judgement b[Γ:=G]: β [x:α[Γ:=G]] is computable.

Proof: Let a:α[Γ:=G] be a c.s.s. for x:α[Γ:=G]; then "G, a:α[Γ:=G]" is a c.s.s. for "Γ, x:α" hence b[Γ,x := G,a] ≡ b[Γ:=G][x:=a]: β[Γ,x := G,a]≡β[Γ:=G][x:=a] is computable.

Observe also that the following fact is an immediate consequence of the definition of computable judgement.

Fact 2.7:

If a:α is derivable, a g and g:α is computable then a:α is computable.

Now we are ready to prove that every derivable judgement is computable; the following lemmas are the inductive steps of a proof by induction on the length of the derivation. Let us begin with the type "arrow".

Lemma 2.8:

If b:β [Γ, x:α] is computable then (λx:α.b):α → β [Γ] is computable.

Proof: Two cases have to be considered:

Γ=Ø: by definition (λx:α.b) (λx:α.b) and b:β [x:α] is computable by hypothesis.

Γ≠Ø: let G be a c.s.s. for Γ; then $b[\Gamma:=G]:\beta[\Gamma:=G]$ $[x:\alpha[\Gamma:=G]]$ is computable, by the head-substitution lemma 2.6, hence the previous case shows that $(\lambda x:\alpha.b)[\Gamma:=G]:\alpha[\Gamma:=G]\to\beta[\Gamma:=G]$ is computable.

Lemma 2.9:

If $c:\alpha\to\beta$ [Γ] and $a:\alpha$ [Γ] are computable then $ap(c,a):\beta$ [Γ] is computable.

Proof: Two cases have to be considered:

Γ=Ø: $ap(c,a)$ has value g if c $(\lambda x:\alpha.b)$ and $b[x:=a]$ has value g; the first statement holds since $c:\alpha\to\beta$ is computable by hypothesis and the last one holds since $b:\beta$ $[x:\alpha]$ is computable by hypothesis and $a:\alpha$ is a c.s.s. for $x:\alpha$; moreover $g:\beta[x:=a]\equiv\beta$ is computable whichever type β is. Hence the conclusion follows by fact 2.7.

Γ≠Ø: let G be a c.s.s. for Γ; then $c[\Gamma:=G]:\alpha[\Gamma:=G]\to\beta[\Gamma:=G]$ and $a[\Gamma:=G]:\alpha[\Gamma:=G]$ are computable judgements hence the previous case shows that $ap(c,a)[\Gamma:=G]:\beta[\Gamma:=G]$ is computable.

Consider now the type "quantification".

Lemma 2.10:

If $b:\beta$ [Γ, X typeVar] is computable then $(\Lambda X.b):\Pi X.\beta$ [Γ] is computable.

Proof: Two cases have to be considered:

Γ=Ø: by definition $(\Lambda X.b)$ $(\Lambda X.b)$ and $b:\beta$ [X typeVar] is computable by hypothesis.

Γ≠Ø: let G be a c.s.s. for Γ; then $b[\Gamma:=G]:\beta[\Gamma:=G]$ [X typeVar] is computable, by the head-substitution lemma 2.5, hence the previous case shows that $(\Lambda X.b)[\Gamma:=G]:\Pi X.\beta[\Gamma:=G]$ is computable.

Lemma 2.11:

If $c:\Pi X.\beta$ [Γ] is computable and α type [Γ] is derivable then $AP(c,\alpha):\beta[X:=\alpha]$ [Γ] is computable.

Proof: Two cases have to be considered:

Γ=Ø: $AP(c,\alpha)$ has value g if c $(\Lambda X.b)$, which holds since $c:\Pi X.\beta$ is computable by hypothesis, and $b[X:=\alpha]$ has value g; also this last statement holds since $b:\beta$ [X typeVar] is computable and the derivable judgement α type is a c.s.s. for X typeVar; moreover $g:\beta[X:=\alpha]$ is computable whichever types α and β are. Hence the conclusion follows by fact 2.7.

Γ≠Ø: let G be a c.s.s. for Γ then $c[\Gamma:=G]:\Pi X.\beta[\Gamma:=G]$ is computable and $\alpha[\Gamma:=G]$ type is derivable hence the previous case shows that $AP(c,\alpha)[\Gamma:=G]:\beta[\Gamma:=G][X:=\alpha[\Gamma:=G]] \equiv \beta[X:=\alpha][\Gamma:=G]$ is computable.

Since the structural rules obviously preserve computability, we have proved that any rule preserves computability and thus we obtain the computability theorem.

Theorem 2.12: (Computability)
Any derivable judgement is computable.

Its main corollary is the normal form theorem.

Corollary 2.13: (Normal form theorem)
If a:α is derivable then a has a value.

References

[Bos-Val] A.Bossi, S.Valentini - *An intuitionistic theory of types with assumptions of high-arity variables* - Annals of Pure and Applied Logic 57 (1992), p.93-149.

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

[MLöf] P.Martin Löf - *Intuitionistic Type Theory* - (Bibliopolis, Napoli 1984).