Proving the Genericity Lemma by Leftmost Reduction is Simple

Jan Kuper

University of Twente, Department of Computer Science
P.O.Box 217, 7500 AE Enschede, The Netherlands
e-mail: jankuper@cs.utwente.nl

Abstract. The Genericity Lemma is one of the most important motivations to take in the untyped lambda calculus the notion of solvability as a formal representation of the informal notion of undefinedness. We generalise solvability towards typed lambda calculi, and we call this generalisation: *usability*. We then prove the Genericity Lemma for *un*-usable terms. The technique of the proof is based on *leftmost* reduction, which strongly simplifies the standard proof.

1 Introduction

In this paper we present an elementary and general proof of the Genericity Lemma. In the untyped lambda calculus this lemma says:

> Suppose M is an unsolvable term, and let N be a normal form.
> If $\boldsymbol{\lambda} \vdash FM = N$, then for all X we have $\boldsymbol{\lambda} \vdash FX = N$.

The informal meaning of this lemma is that if a term M is meaningless (undefined), and if a context containing M is convertible to a well-defined answer, then M did not have any influence on the computation of this answer and so M may be replaced by any term. In fact, the Genericity Lemma is one of the most important motivations to take in the untyped lambda calculus the notion of solvability as a formal representation of the informal notion of undefinedness.

Several proofs of the Genericity Lemma are known. For example, the standard proof (Barendregt 1984, chapter 14.3) uses a topological method, based on the tree topology (using Böhm trees, and showing that the compactification points in this topology are precisely the unsolvable terms). Takahashi gives a simpler proof in the untyped lambda calculus by exploiting the fact that the solvable terms are precisely the terms with a head normal form (Takahashi 1994). In (Kuper 1994, 1995) the Genericity Lemma is proved for a PCF-like calculus by generalising a technique introduced in (Barendregt 1971). This technique requires a tedious analysis of a reduction $FM \twoheadrightarrow N$.

Here, we prove the lemma for a more general situation than just untyped lambda calculus. Hence, we have to generalise the notion of solvability towards

other lambda calculi, for example towards calculi with types, δ-reduction, μ-reduction. This generalisation can not be done directly from the definition of solvability in the untyped lambda calculus.

We will describe a generalisation of solvability (called *usability*) for a PCF-like calculus with product types and list types. We show that the Genericity Lemma holds for *un*usable terms. We prove the lemma by concentrating on the *leftmost* reduction $FM \underset{\ell}{\twoheadrightarrow} N$, which simplifies the proof strongly.

We remark that it is also possible to generalise Takahashi's proof to other calculi. However, this generalisation is not as simple as it might seem at first sight. There are two reasons for this. First, the connection between solvability and head normal forms which holds in the untyped lambda calculus, is in general not true for usability (see section 3).

Second, Takahashi's proof in fact proceeds by induction on the depth of the Böhm tree of a certain term, so a generalisation of this proof requires a generalisation of Böhm trees towards other calculi. This falls outside the scope of this paper.

2 The Lambda Calculus

We prove the Genericity Lemma for a PCF-like calculus, i.e., for a typed lambda calculus with full recursion and some constants, though the proof also works for untyped lambda calculus. We will refer to this calculus by $\boldsymbol{\lambda}$.

We will assume that $\boldsymbol{\lambda}$ has ground types **Nat** and **Bool**. Also, if σ and τ are types, then $\sigma \to \tau$ (functions), $\sigma \times \tau$ (products), and σ^* (lists) are types. The corresponding term formation rules are $\lambda x.M$, $\langle M, N \rangle$, and $[M, N]$ respectively. The final one stands for the cons-operation, where M is of type σ and N is of type σ^*. One might also choose for a constant **cons**, but because of the role of constants in forthcoming definitions (especially definition 2), we do not choose for this option (see section 3). The same holds for a constant `pair` in the case of product types.

Of course, the basic reduction rule of the calculus is the rule of β-reduction:

$$(\lambda x.M)N \to M[x{:=}N]$$

(where $M[x{:=}N]$ denotes substitution of N for all free occurrences of x in M, implicitly avoiding unintended bindings of variables).

The rule of η-reduction

$$\lambda x.Mx \to M$$

(where x is not free in M) may or may not hold.

Since non-termination is essential for the Genericity Lemma, we will assume that full recursion is possible in the calculus. We prefer to represent recursion by means of the μ-abstractor, accompanied with the rule

$$\mu x.M \to M[x{:=}\mu x.M],$$

but clearly, recursion might also be represented by means of fixed point combinators $\mathbf{Y}$.

We assume that the following constants and the accompanying reduction rules (δ-rules) are present in the calculus:

- there are constants for natural numbers, booleans, successor function, test for zero. The accompanying δ-rules are standard,
- there is a conditional `if`, with accompanying rules

$$\texttt{if true} \to \lambda xy.x,$$
$$\texttt{if false} \to \lambda xy.y,$$

- there are constants for the projection functions π_i $(i = 1, 2)$:

$$\pi_i\langle M_1, M_2\rangle \to M_i,$$

and for lists:

$$\texttt{head}\,[M, N] \to M,$$
$$\texttt{tail}\,[M, N] \to N.$$

Notice that all δ-redexes are of the form fM, with f a constant.

Two essential properties of the calculus that are necessary for the proof of the Genericity Lemma given here, are the *Church-Rosser Property* and the *Standardisation Property.*

Church-Rosser property. For all terms M, N_1 and N_2 such that

$$N_1 \twoheadleftarrow M \twoheadrightarrow N_2,$$

there is a term L such that

$$N_1 \twoheadrightarrow L \twoheadleftarrow N_2.$$

A well-known corollary of the Church-Rosser property is that if a term M is convertible to a normal form N, then $M \twoheadrightarrow N$. Of course, $\boldsymbol{\lambda}$ has the Church-Rosser property.

Standardisation. Consider the following reduction:

$$M_0 \xrightarrow{\Delta_0} M_1 \xrightarrow{\Delta_1} \cdots$$

This reduction is a *standard reduction* if there is no $j < i$ such that Δ_i is a residual of a redex in M_j to the left of Δ_j.

Now the standardisation property says: for every $M \twoheadrightarrow N$, there is a standard reduction from M to N. A sufficient and elegant way to prove that a calculus has the standardisation property, is to show that the calculus is a left-normal combinatory reduction system (introduced in (Klop 1980), reformulated with some simplifications in (Kuper 1994)). It is easy to see that indeed $\boldsymbol{\lambda}$ is a left-normal combinatory reduction system.

An important corollary of the standardisation property is the normalisation theorem: if a term M has a normal form N, then $M \twoheadrightarrow N$ by the leftmost reduction, i.e., by contracting the leftmost redex in each reduction step (notation: $M \underset{\ell}{\twoheadrightarrow} N$).

3 Solvability and Usability

We start with the definition of solvability from the untyped lambda calculus, and generalise this notion towards some other calculi (for a more detailed treatment of this generalisation, see (Kuper 1994, 1995)). Recall that solvability is intended to formalise the concept of meaningfulness (or (un)definedness, cf. (Barendregt 1984)).

Definition 1 (Solvability). Let $\lambda\mathbf{x}.M$ be a closure of M. M is *solvable* if there is a sequence of terms $\mathbf{N}$ such that $(\lambda\mathbf{x}.M)\mathbf{N} \twoheadrightarrow \mathbf{I}$, where $\mathbf{I} \equiv \lambda x.x$.

This definition can not be directly generalised towards other calculi for two reasons: because of the restriction to contexts of the form $(\lambda\mathbf{x}.[\_])\mathbf{N}$, and because of the restriction of the result to $\mathbf{I}$. To overcome the first of these restrictions we introduce in $\boldsymbol{\lambda}$ the notion of *strict context*:

Definition 2 (Strict Context). A *strict context* $C[\_]$ is inductively defined as follows:

- the empty context, $[\_]$, is strict
- If $C[\_]$ is a strict context, and M a term, then

  (i) $(C[\_])M$,
  (ii) $\lambda x.C[\_]$

  are strict contexts.
- If f is a constant of function type, and $C[\_]$ is a strict context, then also

  (iii) $f(C[\_])$

  is a strict context.

In this definition we silently assume type correctness. Notice that each strict context has precisely one hole.

Intuitively, the strictness of a context $C[\_]$ means that in a (leftmost) reduction of $C[M]$ information from M is really used. This intuition can be considered as a starting point for a further generalisation of strict contexts towards other calculi.

Motivated by this intuition, we will use strict contexts in order to generalise the notion of solvability towards $\boldsymbol{\lambda}$. To overcome the second restriction in the definition of solvability (definition 1), we allow for any normal form instead of just $\mathbf{I}$. We call our generalisation of solvability: *usability*.

Definition 3 (Usability). A term M is *usable* if there is a strict context $C[\_]$ such that $C[M]$ has a normal form.

The intuition behind this definition is that a term M is meaningful (i.e., usable) if there is a terminating computation in which M is effectively used.

As examples of usable terms we mention that constants are usable. More general, all normal forms are usable. The standard example of a meaningless

term, Ω, is not usable (in $\boldsymbol{\lambda}$ one can define Ω as $\mu x.x$). On the other hand, $\langle x\Omega, \Omega\rangle$ is usable: define the strict context

$$C[\_] \equiv (\lambda x.\boldsymbol{\pi}_1[\_])(\lambda xy.y)\underline{0},$$

and notice that $C[\langle x\Omega, \Omega\rangle] \twoheadrightarrow \underline{0}$.

Now we can motivate our choice to consider pairing as a syntactical term forming construction, and not by means of a constant `pair`: having a constant `pair` would make Ω usable ($\boldsymbol{\pi}_2$(`pair` [_] $\underline{0}$) is a strict context), which is not intended. To avoid this effect, one might add exceptions to definition 2, but this is rather clumsy. Having pairs by means of a syntactical term forming construction is much nicer.

In order to show that usability really is a generalisation of solvability, we restrict definition 2 to the untyped lambda calculus by removing clause (*iii*) from it.

Lemma 4. *In the untyped lambda calculus: M is usable iff M is solvable.*

Proof. Left to the reader. □

As a remarkable difference between the notions of solvability and usability, we mention that the usable terms are *not* precisely the terms with a head normal form. In $\boldsymbol{\lambda}$ one only has that usable terms have a weak head normal form, but not vice versa. In contrast to the situation in untyped lambda calculus, it is highly unlikely that there is a syntactical characterisation of usable terms at all. Consider the following two terms:

$$M_1 \equiv \texttt{if Zero?(Pred } x\texttt{) then } \underline{0} \texttt{ else } \Omega,$$
$$M_2 \equiv \texttt{if Zero?(Succ } x\texttt{) then } \underline{0} \texttt{ else } \Omega,$$

and notice that M_1 is usable, but M_2 is not.

As further evidence for the appropriateness of usability to formalise meaningfulness, we mention that all unusable terms can be consistently identified (respecting their types, of course). If in addition to this identification, a usable term is also identified to the unusable terms, then the calculus becomes inconsistent (Kuper 1994, 1995). Furthermore, usability has the genericity property (see theorem 10).

We will need the following lemma.

Lemma 5. *A is usable iff $\lambda x.A$ is usable.*

Proof. "⇒": If A is usable, then there is a strict context $C[\_]$ such that $C[A]$ has a normal form. Hence, $C[(\lambda x.A)x]$ ($\equiv C[A]$) has a normal form. Since $C[[\_]x]$ is a strict context, it follows that $\lambda x.A$ is usable.

"⇐": If $\lambda x.A$ is usable, then there is a strict context $C[\_]$ such that $C[\lambda x.A]$ has a normal form. Now $C[\lambda x.[\_]]$ is a strict context, so A is usable. □

4 The Genericity Lemma

The Genericity Lemma says: if M is not usable, and FM has a normal form, then for all X, FX has the same normal form.

In the proof of this lemma we will concentrate on the *leftmost* reduction $FM \underset{\ell}{\longrightarrow\!\!\!\!\!\rightarrow} N$, and we show that M is not "in an essential way" involved in a leftmost reduction step. Then it is intuitively clear that M may be replaced by any X without changing the reduction steps.

In order to keep track of M during the reduction, we extend $\boldsymbol{\lambda}$ with a possibility to "pack" M. If M is in an essential way involved in a reduction step, then M must be "unpacked" first.

Definition 6 (The calculus $\boldsymbol{\lambda\Box}$). The *terms* of $\boldsymbol{\lambda\Box}$ are formed by the same term formation rules as the rules of the original calculus $\boldsymbol{\lambda}$. In addition, if M is a $\boldsymbol{\lambda\Box}$-term, then $\boxed{M}$ is a $\boldsymbol{\lambda\Box}$-term.

The *reduction rules* of $\boldsymbol{\lambda\Box}$ are the same as the rules of the original calculus $\boldsymbol{\lambda}$. In addition, there is the $\Box$-rule (the "unpack rule"):

$$\boxed{M} \to M.$$

The term $\boxed{M}$ is *to the left* of all subterms of M.

Substitution is defined in the usual way, extended with the clause

$$\boxed{M}[x{:=}N] \equiv \boxed{M[x{:=}N]}.$$

Strict contexts are defined by precisely the same clauses as in definition 2, and *usability* is defined as in definition 3.

We will use the following notations with the obvious interpretations: $\xrightarrow{\Box}$, $\xrightarrow{\Box}\!\!\!\!\!\rightarrow$, $\underset{\ell}{\xrightarrow{\Box}\!\!\!\!\!\rightarrow}$.

Remarks. If $\boxed{M}$ is involved in a $\boldsymbol{\lambda}$-reduction step in an essential way, then M must first be unpacked. For example, $(\lambda x.\ \boxed{M}\ )N$ and $(\lambda x.M)\ \boxed{N}$ are β-redexes, but $\boxed{\lambda x.M}\ N$ is *not*. To obtain a β-redex from this term, we first have to apply the $\Box$-rule on the subterm $\boxed{\lambda x.M}$. The same holds for terms of the form $\boxed{f}M$ and $f\boxed{M}$, whenever fM is a δ-redex.

Lemma 7. $\boldsymbol{\lambda\Box}$ *has the Church-Rosser property and the standardization property.*

Proof. Straightforward. One may consider $\Box$ as a new constant $\mathbf{c}$ for each type, with the rule

$$\mathbf{c}M \to M,$$

i.e., $\mathbf{c}$ behaves like the identity. It is clear that the resulting calculus is a left-normal CRS. Hence it has the indicated properties (Klop 1980). $\Box$

Lemma 8. $\boxed{M}$ *is usable iff* M *is usable.*

Proof. Immediate from the definition of usability, since $\boxed{M} = M$. □

The following lemma holds in both $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}\Box$.

Lemma 9. *If* $C[M]$ *has a normal form, and the displayed* M *is the leftmost redex in* $C[M]$*, then* M *is usable.*

Proof. Since the displayed M is the leftmost redex in $C[M]$, this M is not inside a box. For the same reason, M is not in the scope of a μ.

Let $C_s[\_]$ be the largest strict context inside $C[\_]$, where the hole of $C_s[\_]$ is the same as the hole of $C[\_]$. It is easily seen that $C[\_]$ is of one of the following four forms:

- $C[\_] \equiv C_s[\_]$. Then $C_s[M]$ has a normal form, i.e., M is usable.
- $C[\_] \equiv C'[F(C_s[\_])]$. Let N be the normal form of $C[M]$, then

$$C'[F(C_s[M])] \underset{\ell}{\longrightarrow\!\!\!\!\!\rightarrow} N \tag{1}$$

  by lemma 7. Since M is the leftmost redex and $C_s[\_]$ is the largest possible strict context, it follows that F is either a variable, or an application term of the form $GL_1 \cdots L_n$ $(n \geq 1)$, with G a constant or a variable. Hence, during reduction (1) reducts of $C_s[M]$ will not be a proper subterm of some redex. Hence, there is an $N' \subseteq N$ such that $C_s[M] \twoheadrightarrow N'$. Clearly, N' is a normal form, so M is usable.
- $C[\_] \equiv C'[\langle C_s[\_], X\rangle]$ or $C[\_] \equiv C'[\langle X, C_s[\_]\rangle]$. We consider the first case, the second case being analogous. Suppose $C'[\langle C_s[M], X\rangle] \twoheadrightarrow N$. Since M is the leftmost redex, it follows that $C_s[M] \twoheadrightarrow N' \subseteq N$. As above, M is usable.
- $C[\_] \equiv C'[\,[C_s[\_], X]\,]$ or $C[\_] \equiv C'[\,[X, C_s[\_]\,]\,]$. As above. □

Theorem 10 (The Genericity Lemma). *Suppose* M *is not usable, and* N *is a normal form. If* $\vdash FM = N$*, then for all* X

$$\vdash FX = N.$$

Proof. The proof consists of two steps:

(a) for closed M only,
(b) for arbitrary M.

Proof of (a): Consider in $\boldsymbol{\lambda}$ the leftmost reduction $\mathcal{R} : FM \underset{\ell}{\longrightarrow\!\!\!\!\!\rightarrow} N$. It is straightforward to construct from $\mathcal{R}$ the leftmost reduction

$$\mathcal{R}' : F\boxed{M} \underset{\ell}{\overset{\Box}{\longrightarrow\!\!\!\!\!\rightarrow}} N.$$

Since M is closed, and since there are no reductions inside boxed terms (the reduction $\mathcal{R}'$ is leftmost) it follows that for all terms of the form $\boxed{L}$ which

arise during the reduction $\mathcal{R}'$, we have that $L \equiv M$. Now suppose there is an application of the $\Box$-rule in $\mathcal{R}'$, say

$$C[\boxed{M}] \xrightarrow{\Box} C[M] \xrightarrow[\ell]{\Box} N.$$

Then $\boxed{M}$ is the leftmost redex in $C[\boxed{M}]$, and so (by lemma 9) $\boxed{M}$ is usable, i.e., M is usable. This is a contradiction, so there are no applications of the $\Box$-rule in $\mathcal{R}'$. Since N is $\Box$-free, it can now easily be shown (by induction on the length of $\mathcal{R}'$) that $\boxed{M}$ can be replaced by any X.

Proof of (b): Let $\mathbf{x}$ be the sequence of variables free in M. Then $\lambda\mathbf{x}.M$ is closed and not usable (by lemma 5). Hence,

$$\begin{aligned} FM &= F((\lambda\mathbf{x}.M)\mathbf{x}) \\ &= (\lambda y.F(y\mathbf{x}))(\lambda\mathbf{x}.M) \\ &= N, \end{aligned}$$

and so

$$\begin{aligned} FX &= F((\lambda\mathbf{x}.X)\mathbf{x}) \\ &= (\lambda y.F(y\mathbf{x}))(\lambda\mathbf{x}.X) \\ &= N. \quad \Box \end{aligned}$$

References

Barendregt, H.P., *Some extensional term models for combinatory logics and lambda calculi*, Ph.D. Thesis, Utrecht.

Barendregt, H.P., *The Lambda Calculus – Its Syntax and Semantics* (revised edition), North-Holland, Amsterdam.

Klop, J.W., *Combinatory Reduction Systems*, Ph.D. Thesis, Mathematical Centre, Amsterdam.

Kuper, J., *Partiality in Logic and Computation – Aspects of Undefinedness*, Ph.D. Thesis, Enschede.

Kuper, J., Usability: formalising (un)definedness in typed lambda calculus, CSL '94, Kazimierz, Poland, 1994. To appear in the proceedings, Springer-Verlag, Berlin, Heidelberg.

Takahashi, M., A simple proof of the Genericity Lemma, *in:* Neil D. Jones, Masami Hagiya, Masahiko Satu (Eds.): *Logic, Language and Computation – Festschrift in Honour of Satoru Takasu*, Springer-Verlag, Berlin, Heidelberg, pp. 117 – 118.