Basic Research in Computer Science

BRICS RS-96-5 M. Goldberg: Gödelisation in the λ-Calculus

Gödelisation in the λ-Calculus

(Extended Version)

Mayer Goldberg

BRICS Report Series **RS-96-5**

ISSN 0909-0878 **March 1996**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

Gödelisation in the λ-Calculus *

Mayer Goldberg
Computer Science Department
Indiana University †
(mayer@cs.indiana.edu)

March 20, 1996

Abstract

Gödelisation is a meta-linguistic encoding of terms in a language. While it is impossible to define an operator in the λ-calculus which encodes all closed λ-expressions, it is possible to construct restricted versions of such an encoding operator modulo normalisation. In this paper, we propose such an encoding operator for proper combinators.

Keywords: Programming Calculi; λ-Calculus; Gödelisation.

1 Prerequisites and Notation

We assume some familiarity with the untyped and simply typed λ-calculi [1, 3]. The set of all terms *generated* by $\{M_1, \ldots, M_n\}$ is $\{M_1, \ldots, M_n\}^+$ [1, Item 8.1.1 (i), Page 165]. The set of all λ-terms is denoted by Λ, the set of all closed λ-terms (combinators) is denoted by Λ^0. When n ranges over the integers, $\ulcorner n \urcorner$ denotes the n-th Church numeral, and when M ranges over all λ-expressions, $\ulcorner M \urcorner$ denotes an encoding of M. The Church successor

*This work was completed while visiting BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation).

†Bloomington, IN 47405, USA.

function is denoted by $\mathbf{Succ}_{\mathsf{Church}}$. The identity combinator is denoted by $\mathbf{I}$. The ordered n-tuple is denoted by $[x_1, \ldots, x_n]$, and the k-th projection function on an n-tuple is denoted by π^n_k. Since the definition of an ordered n-tuple and the respective projection functions play a rôle in the proof of Theorem 3.2, we give their definitions below:

$$\begin{array}{rcll} [x_1, \ldots, x_n] & = & \lambda x.(x\ x_1 \cdots x_n) & \text{The ordered } n\text{-tuple.} \\ \pi^n_k & = & \lambda t.(t\ (\lambda x_1 \cdots x_n.x_k)) & \text{The } k\text{-th projection.} \end{array}$$

2 Introduction

Gödelisation[1] is an effective injection that is used to encode terms in a language[1, Item 6.5.6, Page 143] . It is possible to write a combinator **Gödel** in the λ-calculus, such that

$$(\mathbf{G\ddot{o}del}\ \ulcorner M \urcorner) \;=\; \ulcorner\ulcorner M \urcorner\urcorner. \tag{1}$$

Gödel is the same as Barendregt's Num combinator [1, Item 6.5.9, Page 143]. **Gödel** does not map λ-expressions to their encodings, but rather encodings of λ-expressions to the encodings of their encodings. Indeed, it is impossible to define a combinator that maps λ-expressions to their encodings:

2.1 PROPOSITION: *Gödelisation is necessarily a meta-linguistic notion in the λ-calculus, i.e. there exists no combinator G such that for any closed λ-term M we have*

$$(G\ M) \;=\; \ulcorner M \urcorner \tag{2}$$

Proof: Let $M = (\mathbf{I}\,\mathbf{I})$. We should have $\ulcorner(\mathbf{I}\,\mathbf{I})\urcorner$ different from $\ulcorner \mathbf{I} \urcorner$, but by the Church-Rosser Theorem we have $(G\ (\mathbf{I}\,\mathbf{I})) = (G\ \mathbf{I})$. ∎

In light of Proposition 2.1, we are only interested in encoding λ-terms that have a normal form, and we consider this encoding to be *modulo* the normal form.

But even modulo normalisation, defining a Gödelisation combinator is

[1] Gödelisation takes its name from a proof technique used by Kurt Gödel in his paper "On formally undecidable propositions of Principia Mathematica and related systems" [6].

still a difficult problem, quite different in nature from the combinator we considered in (1). As a milestone on the road to deciding the existence of an encoding combinator such as (2) for all terms modulo normalisation, we consider a weaker notion, that of a *partial Gödeliser:*

2.2 DEFINITION: *A Partial Gödeliser.* Given a set S of combinators, we associate with each $M \in \mathsf{S}$ a λ-expression I_M (which is taken to be "information about M"). A λ-expression G_{S} is said to be a *partial Gödeliser for* S if for each $M \in \mathsf{S}$ we have:

$$(G_{\mathsf{S}}\ M\ I_M) \ = \ \ulcorner M \urcorner \tag{3}$$

A trivial partial Gödeliser might have $\mathsf{S}\ \Lambda^0$, and

$$I_M = \begin{cases} \ulcorner M_{\mathrm{nf}} \urcorner & \text{if } M \text{ has a nf } M_{\mathrm{nf}} \\ \ulcorner \bot \urcorner & \text{if } M \text{ has no nf} \end{cases} \tag{4}$$

The best possible Gödeliser we could hope for has $\mathsf{S} = \Lambda^0$, and $I_M = \ulcorner \bot \urcorner$ (i.e. I_M provides the partial Gödeliser with *no* information), and

$$(G_{\mathsf{S}}\ M\ I_M) \ = \ \begin{cases} \ulcorner M_{\mathrm{nf}} \urcorner & \text{if } M \text{ has nf } M_{\mathrm{nf}} \\ \ulcorner \bot \urcorner & \text{if } M \text{ has no nf} \end{cases} \tag{5}$$

The challenge is to find partial Gödelisers for large and interesting classes of combinators, while keeping the information that needs to be passed on to the partial Gödelisers as simple as possible, so as not to trivialise the task of encoding.

To the best of our knowledge, the only partial Gödeliser in the λ-calculus is due to Berger and Schwichtenberg [2], and encodes simply-typed λ-expressions, given an encoding of their type. Given a simply typed λ-expression M of type $\ulcorner \tau \urcorner$, we have:

$$(G_{\mathrm{st}}\ M\ \ulcorner \tau \urcorner) \ = \ \ulcorner M \urcorner \tag{6}$$

In the next section we derive a partial Gödeliser for the set of all proper combinators. This result is a part of our Ph.D. thesis [8].

3 Gödelisation of Proper Combinators

3.1 DEFINITION: *Proper Combinators* [1, Page 184, Problem 8.5.15], $\mathsf{PC}(n)$. A *proper combinator* of arity n is a λ-expression $\lambda x_1 \cdots x_n.B$ where $B \in \{x_1, \ldots, x_n\}^+$. The set of all proper combinators of arity n is $\mathsf{PC}(n)$.

Note that some proper combinators are not simply typed. For example $(\lambda x.xx)$ has no simple type.

3.2 THEOREM: *There exists G_{PC} such that for any $n \geq 1$ and proper combinator $P \in \mathsf{PC}(n)$ we have:*

$$(G_{\mathsf{PC}}\ P\ \ulcorner n \urcorner) \ = \ \ulcorner P \urcorner$$

Proof: We assume the existence of combinators **Var**, **Abs**, and **App** for encoding variables, abstractions and applications. Specifically:

$$\begin{aligned}
(\mathbf{Var}\ \ulcorner n \urcorner) &= \ulcorner x_n \urcorner \\
(\mathbf{Abs}\ \ulcorner x_n \urcorner\ \ulcorner M \urcorner) &= \ulcorner (\lambda x_n.M) \urcorner \\
(\mathbf{App}\ \ulcorner M \urcorner\ \ulcorner N \urcorner) &= \ulcorner (M\ N) \urcorner
\end{aligned} \tag{7}$$

By defining **Var**, **Abs**, and **App** appropriately, we can obtain encodings of λ-expressions in terms of integers, lists, strings, or any other data structure we might want to work with. For example, in a language such as Scheme we use S-expressions to encode variables, abstractions and applications.

We make use of the following property of the application of two ordered pairs (compare with Barendregt's hint in his text on the λ-calculus, Problem 6.8.15 (ii) [1, Page 149]):

$$\begin{aligned}
([a_1, b_1]\ [a_2, b_2]) &\longrightarrow ((\lambda x.(x\ a_1\ b_1))\ (\lambda x.(x\ a_2\ b_2))) \\
&\longrightarrow ((\lambda x.(x\ a_2\ b_2))\ a_1\ b_1) \\
&\longrightarrow (a_1\ a_2\ b_2\ b_1)
\end{aligned} \tag{8}$$

In particular, we have:

$$([R, a]\ [R, b]) \ = \ (R\ R\ b\ a) \tag{9}$$

By choosing $R = \lambda rba.[r, (\mathbf{App}\ a\ b)]$, we have

$$([R, \ulcorner M \urcorner]\ [R, \ulcorner N \urcorner]) \ = \ [R, \ulcorner (M\ N) \urcorner]. \tag{10}$$

Now pick a proper combinator in $\mathsf{PC}(n)$, $P = \lambda x_1 \cdots x_n.B$, where $B \in \{x_1, \ldots, x_n\}^+$. We obtain $\ulcorner B \urcorner$ as follows:

$$(P\ [R, (\mathbf{Var}\ \ulcorner 1 \urcorner)] \cdots [R, (\mathbf{Var}\ \ulcorner n \urcorner)]) \ = \ [R, \ulcorner B \urcorner] \tag{11}$$

This solves the main problem in defining G_{PC}, i.e., the construction of the body of a proper combinator of arity n. What remains is to wrap encodings of abstractions of the n variables around the encoding of the body. The technique we use is similar to that by Church to derive a definition for the predecessor function[3, Chapter III, §9, Page 31]:

Let

$$\begin{aligned} A_k &= \lambda x.(\mathbf{Abs}\ (\mathbf{Var}\ \ulcorner 1 \urcorner) \\ &\qquad \ddots \\ &\qquad\quad (\mathbf{Abs}\ (\mathbf{Var}\ \ulcorner k \urcorner)\ x) \cdots) && (12) \\ P_k &= (P\, [R, (\mathbf{Var}\ \ulcorner 1 \urcorner)] \\ &\qquad \vdots \\ &\qquad [R, (\mathbf{Var}\ \ulcorner k \urcorner)]) && (13) \end{aligned}$$

The function f maps $[\ulcorner k+1 \urcorner, P_k, A_k]$ to $[\ulcorner k+2 \urcorner, P_{k+1}, A_{k+1}]$. We define f as follows:

$$\begin{aligned} f &= \lambda t.[(\mathbf{Succ}_{\mathsf{Church}}\ (\pi^3_1\ t)), \\ &\qquad (\pi^3_2\ t\ [R, (\mathbf{Var}\ (\pi^3_1\ t))]), \\ &\qquad (\lambda x.(\pi^3_3\ t\ (\mathbf{Abs}\ (\mathbf{Var}\ (\pi^3_1\ t)))))] \end{aligned} \tag{14}$$

The n-th composition of f applied to the triple $[\ulcorner 1 \urcorner, P_0 = P, A_0 = \mathbf{I}]$ reduces to the triple $[\ulcorner n+1 \urcorner, P_n, A_n]$. We get $\ulcorner P \urcorner$ by applying A_n to the second projection of P_n. A definition for G_{PC} is obtained by abstracting over the proper combinator P and the Church numeral n:

$$\begin{aligned} G_{\mathsf{PC}} &= \lambda pn.((\lambda t.(\pi^3_3\ t\ (\pi^2_2\ (\pi^3_2\ t)))) \\ &\qquad (n\ f\ [\ulcorner 1 \urcorner, p, \mathbf{I}])) \end{aligned} \tag{15}$$

We now have for all $n \geq 1$ and for any proper combinator $P = \lambda x_1 \cdots x_n.B \in \mathsf{PC}(n)$. This completes our derivation. ■

4 Conclusion

Proposition 2.1 shows that no Gödeliser for Λ^0 exists, that takes no additional information about the expression it is encoding. Consequently, we consider *partial Gödelisers*, operating on specific subsets of Λ^0 and taking some information about the expressions they are encoding. Berger and Schwichtenberg [2] have constructed a partial Gödeliser which encodes simply-typed λ-expressions, given an encoding of their type. In this paper, we have shown that a partial Gödeliser G_{PC} exists for proper combinators, given their arity.

The fact that Gödelisation is taken modulo the normal form results in a normalisation effect which has been exploited in proof theory [2, Section 7], and in partial evaluation [5].

We have coded the definition for G_{PC} into the programming language Scheme [4], and have used it to visualise the source code (modulo normalisation and α-equivalence) of compiled code. The source code is presented in Appendix A, and a sample run is presented in Appendix B.

Acknowledgements

I am grateful to BRICS[2] for hosting me and for providing a stimulating environment. I would like to thank Prof. Hendrik P. Barendregt for suggesting an improvement to the argument in Proposition 2.1. Thanks are also due to Olivier Danvy, Daniel P. Friedman, Julia L. Lawall, and Larry Moss for their comments and encouragement.

References

[1] Hendrik P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North-Holland, 1984.

[2]Basic Research in Computer Science, Centre of the Danish National Research Foundation.

[2] Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ-calculus. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 203–211, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

[3] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[4] William Clinger and Jonathan Rees (editors). Revised^4 report on the algorithmic language Scheme. *LISP Pointers*, IV(3):1–55, July-September 1991.

[5] Olivier Danvy. Type-directed partial evaluation. In Guy L. Steele Jr., editor, *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Programming Languages*, pages 242 – 257, St. Petersburg Beach, Florida, January 1996. ACM Press.

[6] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[7] Mayer Goldberg. Gödelisation in the λ-calculus. BRICS Research Series RS-95-38, Computer Science Department, Aarhus University, Denmark, July 1995.

[8] Mayer Goldberg. *Recursive Application Survival in the λ-Calculus*. PhD thesis, Department of Computer Science, Indiana University, May 1996. Forthcoming.

A Scheme Code

```
;;; The Identity combinator:

(define I (lambda (x) x))

;;; Routines to facilitate Church-numeral arithmetic:

(define Church-zero (lambda (x) (lambda (y) y)))
(define Church-S+
  (lambda (cn)
    (lambda (x)
      (lambda (y)
        (x ((cn x) y))))))
(define Church-one (Church-S+ Church-zero))
(define integer->Church
  (lambda (n)
    (if (zero? n)
        Church-zero
        (Church-S+ (integer->Church (sub1 n))))))
(define Church->integer (lambda (cn) ((cn add1) 0)))

;;;  The definition of Var, Abs, and App using S-expressions for
;;;  encoding proper combinators:

(define Var
  (lambda (n)
    (string->symbol (format "x~a" (Church->integer n)))))
(define Abs (lambda (v e) (list 'lambda (list v) e)))
(define App (lambda (f x) (list f x)))

;;; Support for ordered pairs:

(define make-pair (lambda (a b) (lambda (s) ((s a) b))))
(define pair->1 (lambda (p) (p (lambda (a) (lambda (b) a)))))
(define pair->2 (lambda (p) (p (lambda (a) (lambda (b) b)))))
```

;;; *Support for ordered triples:*

```
(define make-triple (lambda (a b c) (lambda (s) (((s a) b) c))))
(define triple->1
  (lambda (t) (t (lambda (a) (lambda (b) (lambda (c) a))))))
(define triple->2
  (lambda (t) (t (lambda (a) (lambda (b) (lambda (c) b))))))
(define triple->3
  (lambda (t) (t (lambda (a) (lambda (b) (lambda (c) c))))))
```

;;; *The Gödeliser for proper combinators, from Theorem 3.2*

```
(define Gpc
  (lambda (p n)
    ((lambda (t) ((triple->3 t) (pair->2 (triple->2 t))))
     ((n (lambda (t)
           (make-triple
             (Church-S+ (triple->1 t))
             ((triple->2 t) (make-pair
                              (lambda (v)
                                (lambda (n)
                                  (lambda (m)
                                    (make-pair v (App m n)))))
                              (Var (triple->1 t))))
             (lambda (x)
               ((triple->3 t) (Abs (Var (triple->1 t)) x))))))
      (make-triple Church-one p I)))))
```

B Scheme Session

```
> (load "pc.scm")
```

;;; *Defining the proper combinator* $\lambda xyz.(x\ x\ (y\ y)(y\ y\ (z\ z)))$,
;;; *which is not simply typed:*

```
> (define foo
    (lambda (x)
      (lambda (y)
        (lambda (z)
          (((x x) (y y)) ((y y) (z z)))))))
```

;;; `foo` *denotes a procedure:*

```
> foo
#<procedure foo>
```

;;; *Encoding* `foo` *into a list:*

```
> (Gpc foo (integer->Church 3))
(lambda (x1)
  (lambda (x2)
    (lambda (x3)
      (((x1 x1) (x2 x2)) ((x2 x2) (x3 x3))))))
```

;;; *Encoding is modulo the normal form:*

```
> (Gpc ((lambda (x) (x x)) (lambda (x) x)) (integer->Church 1))
(lambda (x1) x1)
```

Recent Publications in the BRICS Report Series

RS-96-5 Mayer Goldberg. *Gödelisation in the λ-Calculus (Extended Version)*. March 1996. 10 pp.

RS-96-4 Jørgen H. Andersen, Ed Harcourt, and K. V. S. Prasad. *A Machine Verified Distributed Sorting Algorithm*. February 1996. 21 pp. Abstract appeared in *7th Nordic Workshop on Programming Theory*, NWPT '7 Proceedings, 1995.

RS-96-3 Jaap van Oosten. *The Modified Realizability Topos*. February 1996. 17 pp.

RS-96-2 Allan Cheng and Mogens Nielsen. *Open Maps, Behavioural Equivalences, and Congruences*. January 1996. 25 pp. A short version of this paper is to appear in the proceedings of CAAP '96.

RS-96-1 Gerth Stølting Brodal and Thore Husfeldt. *A Communication Complexity Proof that Symmetric Functions have Logarithmic Depth*. January 1996. 3 pp.

RS-95-60 Jørgen H. Andersen, Carsten H. Kristensen, and Arne Skou. *Specification and Automated Verification of Real-Time Behaviour — A Case Study*. December 1995. 24 pp. Appears in *3rd IFAC/IFIP workshop on Algoritms and Architectures for Real-Time Control*, AARTC '95 Proceedings, 1995, pages 613–628.

RS-95-59 Luca Aceto and Anna Ingólfsdóttir. *On the Finitary Bisimulation*. November 1995. 29 pp.

RS-95-58 Nils Klarlund, Madhavan Mukund, and Milind Sohoni. *Determinizing Asynchronous Automata on Infinite Inputs*. November 1995. 32 pp. Appears in Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science: 15th Conference*, FCT&TCS '95 Proceedings, LNCS 1026, 1995, pages 456–471.

RS-95-57 Jaap van Oosten. *Topological Aspects of Traces*. November 1995. 16 pp. To appear in *Application and Theory of Petri Nets: 17th International Conference*, ICATPN '96 Proceedings, LNCS, 1996.