The systematic construction of a one-combinator basis

J. Fokker

RUU-CS-89-14
May 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

The systematic construction of a one-combinator basis

J. Fokker

Technical Report RUU-CS-89-14
May 1989

Department of Computer Science
University of Utrecht
P.O.Box 80.089, 3508 TB Utrecht
The Netherlands

the Systematic Construction of a One-Combinator Basis

Jeroen Fokker
Dept. of Computer Science
University of Utrecht
PObox 80.089
3508 TB Utrecht
the Netherlands

may 1989

Abstract

A single closed λ-expression that generates all λ-expressions is constructed. The derivation is carried out in a systematic way. The resulting basis is simpler than bases known in the literature.

1 Combinator bases

A *combinator* is a closed λ-term, i.e. an expression formed by λ-abstraction and application that does not contain free variables. Some commonly used combinators are

$$\begin{aligned} \mathsf{S} &= \lambda fgx.fx(gx) \\ \mathsf{K} &= \lambda xy.x \\ \mathsf{I} &= \lambda x.x \end{aligned}$$

In this article, lower case is used for variables in the language of λ-calculus, and upper case for variables denoting λ-terms. Fixed expressions are non-italic. Application associates to the left, so ABC means $(AB)C$. Abstractions range as far as possible, so $\lambda x.AB$ means $\lambda x.(AB)$. Multiple abstractions are abbreviated, so $\lambda xy.A$ means $\lambda x.\lambda y.A$.

Combinators are of great importance for the implementation of functional programming languages (see [3]). Functional programs can be translated to an applicative combination of combinators, thus removing the need for λ-abstraction. A set of combinators from which all closed λ-terms (modulo α-, β- and η-conversion) can be constructed using application is called a ***basis***. Curry [2] discovered that finite bases actually exist:

Theorem 1. $\{\mathsf{S}, \mathsf{K}, \mathsf{I}\}$ is a basis.
Proof: Given a λ-expression, eliminate all abstractions by repeated application of the following rules:

$$\begin{aligned} \lambda x.x &\Rightarrow \mathsf{I} \\ \lambda x.A &\Rightarrow \mathsf{K}A \quad \text{if } x \notin \text{Free}(A) \\ \lambda x.AB &\Rightarrow \mathsf{S}(\lambda x.A)(\lambda x.B) \end{aligned}$$

These rules cover all possible cases, since the body of an abstraction is either the bound variable (rule 1), another variable (rule 2), an application (rule 3), or another abstraction (which can be eliminated first). The process terminates as it only introduces abstractions of smaller terms. Rule 1 is sound by the definition of I. Rule 2 is sound by

$$\begin{aligned} &\mathsf{K}A && = \{\text{def. K}\} \\ &(\lambda xy.x)A && = \{\beta\text{-reduction}\} \\ &\lambda y.A && = \{\alpha\text{-conversion, } x \notin \text{Free}(A)\} \\ &\lambda x.A \end{aligned}$$

Rule 3 is sound by

$$\begin{array}{lcl} \mathsf{S}(\lambda x.A)(\lambda x.B) & = & \{\text{def. S}\} \\ (\lambda fgx.fx(gx))(\lambda x.A)(\lambda x.B) & = & \{\beta\text{-reduction (twice)}\} \\ \lambda x.(\lambda x.A)x((\lambda x.B)x) & = & \{\eta\text{-conversion (twice)}\} \\ \lambda x.AB & & \end{array}$$

□

It is well known that the I is not necessary in the basis:

Theorem 2. {S, K} is a basis.
Proof: By theorem 1, {S, K, I} is a basis. Replace all occurrences of I by SKA, where A is an arbitrary term, e.g. K. This rule is sound by

$$\begin{array}{lcl} \mathsf{SK}A & = & \{\text{def. S}\} \\ (\lambda fgx.fx(gx))\mathsf{K}A & = & \{\beta\text{-reduction (twice)}\} \\ \lambda x.\mathsf{K}x(Ax) & = & \{\text{def. K}\} \\ \lambda x.(\lambda xy.x)x(Ax) & = & \{\beta\text{-reduction (twice)}\} \\ \lambda x.x & & \end{array}$$

□

The central problem of this article is to find a single combinator X such that {X} is a basis.

2 A one-combinator basis

We now construct a basis of one combinator. The importance of this basis is mainly theoretic; bases for praactical use tend to be larger rather than smaller than {S, K, I}. The construction is an example of systematic derivation of a solution from its specification. In the process of program transformation, typically some steps are strongly motivated by the context, and some steps are creative design decisions. In the construction below, three design decisions are explicitly identified. We tried to be as little creative as possible. Simplicity is the driving force in the derivation, in order to obtain the most "natural" result.

The goal is to construct a combinator X from which K and S can be constructed by application. Then, by theorem 2, every closed λ-term can be built. Neither of the combinators K and S forms a basis in its own. Therefore both K and S are a combination of more than one X.

As a first design decision we suppose that both are an application whose the lefthand side is a single X. We try and find expressions A and B such that

$$\begin{array}{lcl} \mathsf{X}A & = & \mathsf{K} \\ \mathsf{X}B & = & \mathsf{S} \end{array}$$

Of course A and B have to be distinct. As a second design decision we choose A and B to be the two simplest distinct expressions that can be made with X, and as K is simpler than S we choose A to be simpler than B:

$$\begin{array}{lcl} A & = & \mathsf{X} \\ B & = & \mathsf{XX} \end{array}$$

The specification now reads

$$\begin{array}{lcl} \mathsf{XX} & = & \mathsf{K} \\ \mathsf{X}(\mathsf{XX}) & = & \mathsf{S} \end{array}$$

and by using the first line in the second one

$$\begin{array}{lcl} \mathsf{XX} & = & \mathsf{K} \\ \mathsf{XK} & = & \mathsf{S} \end{array}$$

As X is applied to functions (X and K), it is of the form $\lambda f.M$. Now we must construct the body M out of f. It is not clear where f is to be used in M. But as it is a function, our third design decision is to apply f to something. As f may be instantiated with K, and K has two arguments, we try to apply f to two arguments. Thus we have

$$\mathsf{X} = \lambda f.fPQ$$

We solve P and Q in this equation. This can be done by calculation:

$$\begin{array}{lcl} \mathsf{S} & = & \{\text{specification}\} \\ \mathsf{XK} & = & \{\text{def. X}\} \\ \mathsf{K}PQ & = & \{\text{def. K}\} \\ P & & \end{array}$$

so $P = \mathsf{S}$, and

$$\begin{array}{lcl} \mathsf{K} & = & \{\text{specification}\} \\ \mathsf{XX} & = & \{\text{def. X}\} \\ \mathsf{X}PQ & = & \{\text{def. X}\} \\ PPQQ & = & \{P = \mathsf{S}\} \\ \mathsf{SS}QQ & = & \{\text{def. S}\} \\ \mathsf{S}Q(QQ) & & \end{array}$$

To unfold the definition of S at this point, we need three arguments. We therefore use extensionality, and state that for all A and B

$$\begin{array}{lcl} A & = & \{\text{def. K}\} \\ \mathsf{K}AB & = & \{\text{above}\} \\ \mathsf{S}Q(QQ)AB & = & \{\text{def. S}\} \\ QA(QQA)B & & \end{array}$$

This is clearly fulfilled by $Q = \lambda xyz.x$.

We can summarize this derivation in

Theorem 3. Let $\mathsf{X} = \lambda f.f\mathsf{S}(\lambda xyz.x)$. Then $\{\mathsf{X}\}$ is a basis.
Proof: By theorem 2, $\{\mathsf{S}, \mathsf{K}\}$ is a basis. Replace all occurrences of K by XX and all occurrences of S by X(XX). These substitutions are sound by the derivation above. □

3 Comparison to other bases

In Barendregt [1] some other one-combinator bases are given. The first was found by Meredith in 1963. Some simpler forms were found by Böhm, Barendregt and Rosser. We quote their solutions from [1]:

Meredith	X	=	$\lambda abcd.cd(a(\lambda x.d))$
	U	=	$\mathsf{X}^3\mathsf{X}^2$
	V	=	$\mathsf{X}^4(\mathsf{K}\mathsf{X}^4)$
	K	⇒	$\mathsf{X}^4(\mathsf{X}^4\mathsf{U}\mathsf{X}^2)\mathsf{X}^2$
	S	⇒	$\mathsf{V}(\mathsf{U}(\mathsf{X}^4(\mathsf{X}^4\mathsf{V}^2)(\mathsf{K}\mathsf{X}^4)))$
Böhm	X	=	$\lambda f.f(f\mathsf{S}(\mathsf{K}^3\mathsf{I}))\mathsf{K}$
	K	⇒	XX
	S	⇒	X(XX)
Barendregt	X	=	$\lambda f.f(f\mathsf{S}(\mathsf{KK}))\mathsf{K}$
	K	⇒	XXX
	S	⇒	X(XXX)
Rosser	X	=	$\lambda f.f\mathsf{KSK}$
	K	⇒	XXX
	S	⇒	X(XX)
Fokker	X	=	$\lambda f.f\mathsf{S}(\lambda xyz.x)$
	K	⇒	XX
	S	⇒	X(XX)

To evaluate the simplicity of the various solutions, we define the *size* of an expression to be the number of abstractions and applications in it, and measure the simplicity of a solution by the size of the respective X, K and S. Another criterion for simplicity is the number of normal-order reduction steps it takes to reach normal form, and the (sum of the) sizes of the intermediate expressions during this reduction. Our systematically derived solutions are the smallest in size, and are comparable to Rosser's in normalization speed.

	size			K		S	
	X	K	S	steps	sizes	steps	sizes
Meredith	74	152	380	41	3622	68	17681
Böhm	23	47	71	29	2509	63	7792
Barendregt	18	56	75	24	1803	53	5923
Rosser	14	44	44	8	262	11	316
Fokker	12	25	38	9	167	12	352

4 Another calculation

Rosser's solution can be calculated from its specification in a similar way as we did in section 2. The specification reads (note the extra X in the first line):

$$\begin{aligned} \mathsf{XXX} &= \mathsf{K} \\ \mathsf{X(XX)} &= \mathsf{S} \end{aligned}$$

A sufficient condition for $\mathsf{XXX} = \mathsf{K}$ is $\mathsf{XX} = \mathsf{KK}$ (in fact this is the condition for $\mathsf{XX}X = \mathsf{K}$). As before we use the first line in the second one to get

$$\begin{aligned} \mathsf{XX} &= \mathsf{KK} \\ \mathsf{X(KK)} &= \mathsf{S} \end{aligned}$$

Note that $\mathsf{KK}ABC = B$, so KK needs three arguments. Therefore in this construction we suppose X to be of the form

$$\mathsf{X} = \lambda f.fPQR$$

As before we calculate by $\mathsf{S} = \mathsf{X(KK)} = \mathsf{KK}PQR = Q$ and $\mathsf{KK} = \mathsf{XX} = \mathsf{X}PQR = PPQRQR = PP\mathsf{S}R\mathsf{S}R$ that

$$\begin{aligned} Q &= \mathsf{S} \\ PP\mathsf{S}R\mathsf{S}R &= \mathsf{KK} \end{aligned}$$

We are left with one equation in two unknowns (P and R). It has many solutions, some of which can be found easily. For example:

- $P = \lambda abcde.e$ and $R = \mathsf{KK}$
- $P = \lambda abcde.c$ and $R = \mathsf{KK}$
- $P = \lambda abcde.ce$ and $R = \mathsf{K}$
- $P = \mathsf{K}$ and $R = \mathsf{K}$

Only the last one is not completely obvious. It is justified by $\mathsf{KKSKSK} = \mathsf{KKSK} = \mathsf{KK}$. This solution yields Rosser's $\mathsf{X} = \lambda f.f\mathsf{KSK}$. It was the fact that there is one equation with two unknowns that motivated us to look for a simpler X.

References

[1] Barendregt, H.P.: *The lambda calculus, its syntax and semantics.* North-Holland, 1984.

[2] Curry, H.B. and R. Feys: *Combinatory logic* Vol. I. North-Holland, 1958.

[3] Peyton Jones, S.L.: *The implementation of functional programming languages.* Prentice-Hall, 1987.