A one-element basis for the SKI logical combinators

tslil clingman

6^{th} May, 2014

1. Introduction

These notes were distributed as part of my first oral presentation during my honours year at the University of Cape Town. Students were required to perform some independent study and present on a topic of their choosing. I chose to learn about bases for the logical combinators, and in so doing observed a straightforward construction which witnessed that there are countably many one-combinator bases. At the time i claimed attribution for this construction, in the interleaving years i have become convinced that this was already known. Nevertheless, to preserve the original document i have not altered this claim. That is, what follows are the original notes, edited only for formatting.

2. Background

We will assume some familiarity with the untyped lambda calculus and logical combinators, but nevertheless recall some details below.

2.1. The untyped λ-calculus

Definition 2.1. Given an alphabet of variables Σ we inductively define the set of λ-expressions, denoted $\Lambda(\Sigma)$, as follows.

1. $x \in \Sigma \Rightarrow x \in \Lambda(\Sigma)$
2. $x \in \Sigma, A \in \Lambda \Rightarrow (\lambda x.A) \in \Lambda(\Sigma)$
3. $A, B \in \Lambda \Rightarrow (AB) \in \Lambda(\Sigma)$

The second and third inductive rules above are usually referred to as *abstraction* and *application* respectively.

Remark 2.2 (Notation). For brevity and ease of use, the following conventions are employed.

- Elements of Σ and Λ are denoted by lower- and upper-case Roman letters respectively.
- Sequential abstractions are contracted, $(\lambda x.(\lambda y.(\mu))) = (\lambda xy.(\mu))$.
- Left-associative application, $ABC = ((AB)C)$.
- Parentheses elision $(AB) = AB$.
- The body of an abstraction extends as far right as possible, $\lambda x.AB = (\lambda x.(AB))$.

Definition 2.3 (Free and bound variables)**.** The set of free variables for a term $A \in \Lambda(\Sigma)$ is inductively defined as follows.

1. $\text{Free}\{x\} = \{x\}$
2. $\text{Free}(\lambda x.A) = \text{Free}(A) \setminus \{x\}$
3. $\text{Free}(AB) = \text{Free}(A) \cup \text{Free}(B)$

A term A is said to be *closed* when $\text{Free}(A) = \phi$. A variable x which appears in a term A but for which $x \notin \text{Free}(A)$ is said to be *bound*.

Example 2.4. Let $A = \lambda x.y$ and $B = \lambda z.Az = \lambda zx.yz$, then $\text{Free}(A) = \text{Free}(B) = \{y\}$.

Definition 2.5 (Reduction)**.** In applying terms to one another, we may seek to reduce the result without changing its "meaning". This may be achieved through the application of the following three reductions.

(α) Bound variables may be renamed providing nothing "bad" happens.

(β) $(\lambda x_1 \dots x_{n+1}.A)B$ may be reduced to $(\lambda x_1 \dots x_n.C)$, where C is A with the "appropriate" substitution of B for x_{n+1} in A.

(η) $\lambda x.Ax$ may be reduced to A when $x \notin \text{Free}(A)$.

We will abuse notation and write $A \overset{\alpha}{=} B$ and so on when the term B is obtained from A by α reduction.

Example 2.6. $\lambda x.x \overset{\alpha}{=} \lambda y.y$, $(\lambda abcd.dbaac)leoh \overset{\beta}{=} hello$, and $\lambda x.(\lambda y.yy)x \overset{\eta}{=} \lambda y.yy$.

2.2. Logical combinators

Definition 2.7 (Combinator)**.** A logical combinator is a closed term. The three combinators of interest are given by $S = \lambda fgx.(fx)(gx)$, $K = \lambda ab.a$, and $I = \lambda x.x$.

Remark 2.8. Some other common combinators are

1. $B = \lambda abc.a(bc)$
2. $C = \lambda abc.acb$
3. $W = \lambda ab.abb$
4. $T = \lambda ab.ba$
5. $M = \lambda a.aa$
6. $J = \lambda abcd.(ab)(adc)$

Definition 2.9 (Span and basis)**.** Fixing an alphabet Σ, the span of a set S of combinators over Σ is a set Ω defined inductively through $A \in S \Rightarrow A \in \Omega$ and $A, B \in \Sigma \Rightarrow AB \in \Omega$. The set S is said to form a basis if every closed term of $\Lambda(\Sigma)$ is in its span Ω.

Theorem 2.10 (Curry's Algorithm)**.** *The combinators $\{S, K, I\}$ form a basis.*

Proof. We give an algorithm to transform a λ-expression into a combinator, using just $\{S, K, I\}$. We then show its correctness, totality, and that it terminates. Given a λ-expression we repeatedly apply the following transformation rules to the term, beginning with the deepest structures.

1. $\lambda x.x \to I$
2. $\lambda x.A \to KA,\ x \notin \text{Free}(A)$
3. $\lambda x.Ax \to A,\ x \notin \text{Free}(A)$
4. $\lambda x.AB \to S(\lambda x.A)(\lambda x.B)$

The rules are correct due to the following reductions.

1. $I \stackrel{\text{def}}{=} \lambda x.x$, $A \stackrel{\eta}{=} \lambda x.Ax$, and $KA \stackrel{\text{def}}{=} (\lambda xy.x)A \stackrel{\beta}{=} \lambda y.A \stackrel{\alpha}{=} \lambda x.A$
2. $S(\lambda x.A)(\lambda x.B) \stackrel{\text{def}}{=} (\lambda fgy.(fy)(gy))(\lambda x.A)(\lambda x.B) \stackrel{\beta}{=} \lambda y.((\lambda x.A)y)((\lambda x.B)y) \stackrel{\eta\alpha}{=} \lambda x.AB$

Note that the body of a λ-expression is precisely one of the following: a bound variable, a free variable, an application, or an abstraction. The first three correspond neatly to the first four rules in the algorithm, while the fourth encompasses the recursive step. Finally, to see that the algorithm terminates notice that, where λ-expressions are introduced, they are of smaller terms. □

Example 2.11. Let $A = \lambda xy.yx$, applying Curry's Algorithm gives

$$\begin{aligned}\lambda xy.yx =& \lambda x.\lambda y.yx \stackrel{4}{=} \lambda x.(S(\lambda y.y)(\lambda y.x)) \stackrel{1}{=} \lambda x.(SI(\lambda y.x)) \stackrel{2}{=} \lambda x.(SI(Kx)) \\ &\stackrel{4}{=} S(\lambda x.SI)(\lambda x.Kx) \stackrel{2}{=} S(K(SI))(\lambda x.Kx) \stackrel{3}{=} S(K(SI))K\end{aligned}$$

Proposition 2.12. *$I = SKA$, for any combinator A.*

Proof. $SKA \stackrel{\text{def}}{=} (\lambda fgy.(fy)(gy))KA \stackrel{\beta}{=} \lambda y.(Ky)(Ay) \stackrel{\text{def}}{=} \lambda y.((\lambda ab.a)y)(Ay) \stackrel{\beta}{=} \lambda y.y \stackrel{\text{def}}{=} I$ □

Corollary 2.13. *$\{S, K\}$ is a basis.*

3. One-element basis

Given that $\{S, K\}$ is a basis, we need only find a combinator whose self-application in some combination yields S and K in order for it to serve as a one-element basis. Let L be this mystery combinator, we desire $LA = K$ and $LB = S$. However, both A and B must be some self-application of L and so, because K is simpler than S we prescribe $LL = K$ and $L(LL) = S$ to find

$$LL = K, \quad LK = S$$

Now we note that L must accept a function as its first argument (L and K) and so must be of the form $L = \lambda f.A$. Given that f could be K, and $K = \lambda xy.x$ we elect to apply f to two arguments

$$L = \lambda f.fAB$$

But what are A and B? Well, $LK = S$ but $LK = KAB = A$ so $A = S$ and

$$L = \lambda f.fSB$$

What of B? We attempt to use the only other equation we have, viz., $LL = K$ which gives

$$LL = LSB = SSBB = SB(BB) = K$$

In order to solve this for B, we apply $(LL)PQ$

$$(LL)PQ = SB(BB)PQ = (BP)(BBP)Q \stackrel{\text{req}}{=} KPQ = P$$

Certainly a solution to this equation is given by the obvious choice $Q = \lambda xyz.x = S(KK)K$ and so, using Curry's Algorithm and the result for A, we may state that

$$L = \lambda f.fS(\lambda xyz.x) = S(S(SKK)(KS))(K(S(KK))K)$$

is a one-element basis.

3.1. Further constructions

Definition 3.1. The constant A-valued combinator on n parameters, for $n \in \mathbb{N}$, is

$$A_n = \underbrace{K(\cdots(K}_{n} A)\cdots)$$

Theorem 3.2. *There are at least countably many one-element bases.*

Proof. By construction, fix $n \geq 2$ and set

$$L = \lambda f.f \underbrace{I \ldots I}_{n-1} S(K_n)_n$$

It is straightforward to verify that $L^{n+1} = K$ and $L(LL) = S$. □

Author	Combinator (L)	K	S
Barendregt[1]	$\lambda f.f(fS(KK))K$	LLL	$L(LLL)$
Böhm[1]	$\lambda f.f(fS(KI))K$	LL	$L(LL)$
Rosser[1]	$\lambda f.fKSK$	LLL	$L(LL)$
Fokker[2]	$\lambda f.fS(\lambda xyz.x)$	LL	$L(LL)$
clingman	For $n \geq 2$, $\lambda f.f \underbrace{I \ldots I}_{n-1} S(K_n)_n$	L^{n+1}	$L(LL)$

Table 1: Alternative one-element basis combinators

Author	Combinator in terms of S, K, I
Barendregt	S (S I (S (S I (K S)) (K (K K)))) (K K)
Böhm	S (S I (S (S I (K S)) (K (K I)))) (K K)
Rosser	S (S (S I (K K)) (K S)) (K K)
clingman	S (S ($\cdots$(S I (K I))$\cdots$(K I)) (K S)) (K (K_n)$_n$))

Table 2: Elaborated one-element basis combinators

References

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

[2] J Fokker, "The systematic construction of a one-combinator basis". Technical Report RUU-CS-89-14, Dep. Comp. Sci., Uni. Utrecht, 1989.