From L to $\lambda\beta$

Gert Smolka, Saarland University

December 4, 2017

We discuss the λ-calculus $\lambda\beta$ assuming that the reader is familiar with the call-by-value λ-calculus L and abstract $\lambda\beta$.

1 Basics

$\lambda\beta$ may be seen as a generalisation of L where equivalence $s \equiv t$ and reduction $s \succ t$ are meaningful for open terms. Even for closed terms the two systems are different since in $\lambda\beta$ every redex $(\lambda x.s)t$ can be replaced with βst while in L the term t must be an abstraction and the redex must not be within an abstraction. In addition, β in $\lambda\beta$ is quite different from β in L.

What $\lambda\beta$ and L have in common is the type of terms. Recall that we employ de Bruijn terms that we write informally as Church terms. Moreover, if we have $s \succ t$ in L and s is closed, we also have $s \succ t$ in $\lambda\beta$. The converse is not true.

$\lambda\beta$ is best understood as an equational logic, so it is helpful to see equivalence $s \equiv t$ as the primary notion and reduction $s \succ t$ as the secondary notion.

Term equivalence and reduction in $\lambda\beta$ are generated by

$$(\lambda x.s)t \equiv \beta st \qquad \text{and} \qquad (\lambda x.s)t \succ \beta st$$

where t can be any term and the function β is different from the function used for L. Moreover, replacement can take place anywhere; for instance, we have $\lambda x.II \succ \lambda x.I$.

You can get intuitions for $\lambda\beta$ from Coq, where β-conversion is realized as in $\lambda\beta$. We may say that $\lambda\beta$ realizes equivalence and reduction as required logically, while L realizes reduction as required by call-by-value functional programming.

Irreducibility is different in $\lambda\beta$ and L. We call a term of the form $(\lambda x.s)t$ a **β-redex**, and say that a term is **β-normal** if it contains no β-redex.

Fact 1 A term is irreducible in $\lambda\beta$ if and only if it is β-normal.

Theorem 2 In $\lambda\beta$, reduction is confluent and the Church-Rosser property holds for equivalence and reduction.

Proof The proof has been done for abstract $\lambda\beta$. What remains to be done is to define β and show that is compatible with parallel reduction. ■

We define

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

Note that $K = \mathrm{F}$. We use K for purposes that are unrelated to booleans. The following fact states important facts about reduction and equivalence in $\lambda\beta$ that are not true for L.

Fact 3 For all terms s, t, and u the following reductions are valid.

$$\begin{aligned} (\lambda x.s)x &\succ s \\ (\lambda x.s)t &\succ s && \text{if } x \text{ not free in } s \\ Kst &\succ^2 s \\ Sstu &\succ^3 su(tu) \end{aligned}$$

When we write $Kst \succ (\lambda x.s)t \succ s$, we can see a crucial point: The variable x must be chosen such that it is not free in s. If terms are formalized as Church terms, β will have to rename bound variables (e.g., $(\lambda xy.x)y \succ \lambda z.y$).

2 SK-Normal Form

We call a term an **SK-term** if it can be obtained with variables, K, S, and applications.[1] It turns out that in $\lambda\beta$ every term is equivalent to an SK-term.

Fact 4

$$\begin{aligned} \lambda x.x &\equiv SKK \\ \lambda x.s &\equiv Ks && \text{if } x \text{ not free in } s \\ \lambda x.st &\equiv S(\lambda x.s)(\lambda x.t) \end{aligned}$$

Theorem 5 Every term is equivalent to an SK-term.

Proof From Fact 4 it is clear that for every SK-term s the abstraction $\lambda x.s$ is equivalent to an SK-term (follows by induction on s). It now follows that every term s is equivalent to an SK-term (again by induction on s). ■

[1] Example and counterexample: SKK is an SK-term and I is not an SK-term.

3 Recursion Operator

In $\lambda\beta$ it is easy to give a term that can serve as recursion operator.

Fact 6 Let $C := \lambda fg.g(ffg)$ and $R := CC$. Then $Rs \succ^2 s(Rs)$ for all terms s.

Think of C and f as *copy term* and of g as *template*.

Note that R has no normal form. Hence no term containing R has a normal form. Thus recursive procedures don't have normal forms. This is in contrast to L, where recursive procedures are normal (with respect to reduction in L).

The recursion operator for L can be obtained as a refinement of the recursion operator for $\lambda\beta$.

Fact 7 Let $C := \lambda fg.g(\lambda x.ffgx)$ and $\rho s := \lambda x.CCsx$. Then $(\rho u)v \succ^3 u(\rho u)v$ for all procedures u and v in L.

4 Church Numerals

In $\lambda\beta$, numbers can be represented as β-normal procedures in such a way that the usual arithmetic operations can be realized without recursion. The technique is due to Church and represents a number n as a procedure $\lambda fa.f^n a$ iterating a given function n times on a given value. Here is the formal definition:

$$\begin{aligned} \overline{n} &:= \lambda fa.f^n a \\ f^0 s &:= s \\ f^{n+1} s &:= f(f^n s) \end{aligned}$$

Note the use of the auxiliary function $s^n t$. We call the term $\overline{n}$ the **Church numeral for n**. Here are the first four Church numerals.

$$\begin{aligned} \overline{0} &= \lambda fa.a \\ \overline{1} &= \lambda fa.fa \\ \overline{2} &= \lambda fa.f(fa) \\ \overline{3} &= \lambda fa.f(f(fa)) \end{aligned}$$

Note that the Church numerals are β-normal procedures. This ensures that numerals for different numbers are not β-equivalent.

Sometimes it is helpful to think of a numeral $\overline{n}$ as a procedure that applied to a function f yields the function f^n.

Here are procedures for successors, addition, and multiplication:

$$\begin{aligned}
\mathsf{succ} &:= \lambda x f a. f(x f a) \\
\mathsf{add} &:= \lambda x. x\, \mathsf{succ} \\
\mathsf{mul} &:= \lambda x y. x(\mathsf{add}\, y)\, \overline{0}
\end{aligned}$$

Fact 8 $\mathsf{succ}\, \overline{n} \;\equiv\; \overline{n+1}$.

Proof $\mathsf{succ}\, \overline{n} \;\succ\; \lambda f a.\ f(\overline{n} f a) \;\succ^2\; \lambda f a.\ f(f^n a) = \lambda f a.\ f^{n+1} a = \overline{n+1}$. ■

Note that the equality steps account for the auxiliary function. Also note that the proof cannot be done in L because the second reduction is done within an abstraction. It seems that a procedure computing successors of Church numerals cannot be defined in L. On the other hand, the operations for Scott numerals work both in L and $\lambda\beta$.

Fact 9 $\mathsf{add}\, \overline{m}\, \overline{n} \;\equiv\; \overline{m+n}$.

Proof By induction on m.
For $m = 0$ we have $\mathsf{add}\, \overline{0}\, \overline{n} \;\succ\; \overline{0}\, \mathsf{succ}\, \overline{n} \;\succ^2\; \overline{n}$.
For $m = Sm$ we have

$$\begin{aligned}
\mathsf{add}\, \overline{Sm}\, \overline{n} &\succ \overline{Sm}\, \mathsf{succ}\, \overline{n} \\
&\succ^2 \mathsf{succ}^{Sm}\, \overline{n} \\
&= \mathsf{succ}\, (\mathsf{succ}^{m}\, \overline{n}) \\
&\prec^2 \mathsf{succ}\, (\overline{m}\, \mathsf{succ}\, \overline{n}) \\
&\prec \mathsf{succ}\, (\mathsf{add}\, \overline{m}\, \overline{n}) \\
&\equiv \mathsf{succ}\, \overline{m+n} && \text{by inductive hypothesis} \\
&\equiv \overline{Sm+n} && \text{by Fact 8}
\end{aligned}$$

■

Note that the proof of Fact 9 uses bidirectional reasoning. Since reduction in $\lambda\beta$ is confluent and $\overline{m+n}$ is normal, Fact 9 implies via Church-Rosser the reduction $\mathsf{add}\, \overline{m}\, \overline{n} \;\succ^*\; \overline{m+n}$.

Fact 10 $\mathsf{mul}\, \overline{m}\, \overline{n} \;\equiv\; \overline{m \cdot n}$.

Proof By induction on m using Fact 9. ■