Turing's contributions to lambda calculus

Henk Barendregt[a], Giulio Manzonetto[a,1]

[a]*Radboud University, Intelligent Systems, Nijmegen, The Netherlands*

1. Fixed point combinators in untyped lambda calculus

The untyped lambda calculus was introduced in 1932 by Church as part of an investigation in the formal foundations of mathematics and logic. The two primitive notions of the lambda calculus are *application* and λ-*abstraction*. Application, written MN, is the operation of applying the term M considered as an algorithm to the term N considered as an input. Lambda abstraction, written $\lambda x.M$, is the process of forming a function from the expression M (possibly) depending on x. We refer to Barendregt et al. (2012) (this volume) for an intuitive account of the system.

An important feature of lambda calculus is that it has *fixed point combinators*, namely programs Y satisfying $YM = M(YM)$ for all M's. These constitute the main ingredient for writing recursive programs in functional style. Turing contributed to this subject by providing a fixed point combinator Θ having the additional property that the equality between ΘM and $M(\Theta M)$ results simply by reducing the former to the latter (which is not the case, in general). Finally we report how Böhm and van der Mey gave a general receipt to generate many new fixed point combinators starting from a fixed one. Turing's fixed point operator can be obtained in this way.

Lambda terms, reduction and conversion

Formally, the set Λ of λ-*terms* is defined inductively as follows.
every variable x is in Λ;
if $M, N \in \Lambda$, then $MN \in \Lambda$;
if $M \in \Lambda$, then $\lambda x.M \in \Lambda$, for every variable x.

Email addresses: `henk@cs.ru.nl` (Henk Barendregt), `g.manzonetto@cs.ru.nl` (Giulio Manzonetto)

[1]Supported in part by NWO Project 612.000.936 CALMOC.

Lambda abstraction is a 'binder', therefore a variable x in M is called *bound* if it occurs in the scope of a 'λx', and is called *free* otherwise. As usual we consider λ-terms up to *α-conversion*, i.e. we consider equal those λ-terms only differing for the names of their bound variables. For example $\lambda x.x = \lambda y.y$.

The *β-reduction*, which specifies how λ-terms compute, is defined as the contextual closure of the following rule:

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

where $M[x{:=}N]$ denotes the result of substituting the term N for every occurrence of x in M, subject to the usual proviso about renaming bound variables in M to avoid capture of free variables in N. A term of the shape $(\lambda x.M)N$ is called *redex* and $M[x{:=}N]$ is its *contractum*.

Multistep β-reduction is denoted by $\twoheadrightarrow_\beta$. This means that $M \twoheadrightarrow_\beta N$ iff $M = M_0 \to_\beta \cdots \to_\beta M_n = N$ for some $n \geq 0$. The *β-conversion*, written $M =_\beta N$, is the equivalence relation generated by $\to_\beta$ (i.e., its reflexive-symmetric-transitive closure). The Church-Rosser theorem in lambda calculus states that

$$M =_\beta N \text{ iff } M \text{ and } N \text{ have a common reduct.}$$

Fixed points

Despite the fact that its syntax is very simple, the lambda calculus is a Turing complete programming language. One aspect of the richness of its expressive power is the presence of fixed points that allow to write recursive programs.

Theorem 1.1 (Fixed Point Theorem)**.** *For all λ-terms F there is a λ-term M such that $FM =_\beta M$.*

PROOF. Take $M \triangleq \omega_F\omega_F$, where $\omega_f \triangleq \lambda x.f(xx)$. Then

$$M \triangleq \omega_F\omega_F =_\beta F(\omega_F\omega_F) \triangleq FM. \quad \square$$

In fact one can show that a fixed point for F can be found uniformly, that is by a term that takes F as input. If a λ-term Y satisfies $YF =_\beta F(YF)$ for all $F \in \Lambda$, then is called a *fixed point combinator* (fpc).

Corollary 1.2 (Curry)**.** *Let $\mathsf{Y}{\triangleq}\lambda f.\omega_f\omega_f$. Then Y is a fixed point combinator.*

The term Y also is called the *paradoxical combinator*, as it abstracts the argument in Russell's paradox.

An fpc Y is *reducing* if for all $M \in \Lambda$ one has $YM \twoheadrightarrow_\beta M(YM)$. This notion is useful as in many applications one needs for a fixed point M of F that $M \twoheadrightarrow_\beta FM$. It is easy to check that Y is not reducing. Therefore one cannot take $M \triangleq \mathsf{Y}F$ to get $M \twoheadrightarrow_\beta FM$.

In Turing (1937) a more convenient fpc is constructed that is reducing.

Proposition 1.3 (Turing)**.** *There exists a reducing fpc.*

Proof. Define $\Theta \triangleq AA$, where $A \triangleq \lambda xy.y(xxy)$. Then one has

$$\Theta F \triangleq AAF \triangleq (\lambda xy.y(xxy))AF \to_\beta (\lambda y.y(AAy))F \to_\beta F(\Theta F). \quad \Box$$

The next lemma shows how Θ arises naturally.

Lemma 1.4 (Böhm, van der Mey)**.** *A term Y is an fpc if it is a fixed point of the peculiar term $\delta = \lambda xy.y(xy)$.*

Proof. If $Y = \delta Y$, then $YF = \delta YF = F(YF)$. $\Box$

For the converse, see Barendregt (1984), Lemma 6.5.3.

From Lemma 1.4 it follows that, starting from a given fpc Y, one can derive an infinite sequence of fpc's.

$$Y_0 \triangleq Y, \qquad Y_{n+1} \triangleq Y_n\delta.$$

A natural question is whether all these fpc's are different. In Endrullis et al. (2010) it is proved, using 'clocked Böhm-trees', that starting from Curry's fpc Y there are no duplicates in the sequence $\mathsf{Y}_0, \mathsf{Y}_1, \mathsf{Y}_2, \cdots$ (the Böhm sequence). The problem is open for sequences starting from an arbitrary fpc Y. Note that Turing's fpc occurs in the Böhm sequence: $\Theta =_\beta \mathsf{Y}_1$, as

$$\mathsf{Y}_1 \triangleq \mathsf{Y}\delta \to_\beta (\lambda x.\delta(xx))(\lambda x.\delta(xx)) \to_\beta (\lambda xy.y(xxy))(\lambda xy.y(xxy)) \triangleq \Theta.$$

2. Weak Normalization of simply typed lambda calculus

Both for programming and theory the property of normalization is crucial. Termination can be seen as an issue of program correctness. The problem of finding all possible inhabitants of a given type in the simply typed lambda calculus relies on the fact that all typable terms have a normal form. An early proof of this weak normalization result for the simply typed lambda calculus is due to Turing and is published posthumously in Gandy (1980). The idea of the proof, is to find a reduction strategy and a measure function mapping terms into some well-founded set, such that the measure is strictly decreasing throughout steps in the computation.

One can see nicely from the notes how Turing wrote them informally, for his own use (as we all start doing). He states that he well-orders the terms, but uses a map f to multisets with the multiset order (which indeed is a well-ordering of type ω^ω), but as the map is not injective, there is no ordering on terms. Then he states—like thinking aloud—that if $M \to_\beta N$ by reducing a redex of highest order, then $f(M) > f(N)$, which is not quite correct. Turing then adds "this at any [rate] will be the case, if we choose the unreduced part of highest order whose λ lies furthest to the right." This indeed yields weak normalization.

The rest of this section is devoted to give a sketch of the technical proof; the reader not interested in technicalities can skip it until Theorem 2.4.

Simply typed lambda terms and reduction

Simply typed λ-calculus, introduced by Church in 1940, is a refinement of λ-calculus where λ-terms are decorated with suitable *simple types*. Such types are called 'simple' since they are built up from atomic types $\mathbb{A}$ using only the arrow constructor $\to$. We will write $\mathbb{T}$ for the set of all simple types.

In the process of building a λ-term M of type σ, written M^σ, we need to check that all types of his subterms are compatible with each other. For a variable x there are no constraints, therefore we have x^σ for all simple types σ. The λ-term $\lambda x.M$ represents a function, therefore it will have an arrow type $\sigma \to \tau$, which is possible only if M^τ and x^σ; we will then write $(\lambda x^\sigma.M^\tau)^{\sigma\to\tau}$. The λ-term MN represents the application of M seen as a function to N seen as an argument; therefore we need that the type of M is an arrow $\sigma \to \tau$ and the type of N is σ; this will be written $(M^{\sigma\to\tau}N^\sigma)^\tau$.

We denote the set of all simply typed λ-terms by Λ^{st}. For instance we have $(\lambda x^\alpha.x^\alpha)^{\alpha\to\alpha} \in \Lambda^{\text{st}}$ and $(\lambda x^\alpha.\lambda y^\tau.x^\alpha)^{\alpha\to\tau\to\alpha} \in \Lambda^{\text{st}}$. When no confusion can arise, we will omit some type annotations and write, e.g., $\lambda x^\alpha.x$.

It is clear that not all λ-terms can be typed in this system. For example $\lambda x.xx$ cannot be typed. Suppose indeed that $\lambda x^\sigma.x^\tau x^\rho$ is a valid typing for some σ, τ, ρ. Then τ must be an arrow type $\tau_1 \to \tau_2$, since the corresponding x is in functional position, then $\rho = \tau_1$ since the corresponding x is in argument position and σ should be equal both to $\tau_1 \to \tau_2$ and to τ_1, which is impossible.

In the simply typed λ-calculus the β-reduction has the same shape as in the untyped case

$$(\lambda x^\sigma.M)N \to_\beta M[x{:=}N],$$

but it can only be applied to well-typed terms. We say that a term $M \in \Lambda^{\mathrm{st}}$ is in *β-normal form*, if there is no $N \in \Lambda^{\mathrm{st}}$ such that $M \to_\beta N$.

The proof of weak normalization

We will now prove that starting from an arbitrary $M \in \Lambda^{\mathrm{st}}$ there is a reduction strategy, specifying at every step what redex should be contracted, so that M reaches a normal form. This is called weak normalization.

Following Turing in Gandy (1980) we define a measure $|\cdot| : \Lambda^{\mathrm{st}} \to \omega^2$ mapping every simply typed λ-term into an element of ω^2, which can be seen as the well-founded set $\mathbb{N} \times \mathbb{N}$ lexicographically ordered.

The idea is that the length $\ell(R)$ of a redex $R = (\lambda x^\sigma.M^\tau)^{\sigma\to\tau}N^\sigma$, which is an estimate of its 'complexity', is given by the length of its arrow type $\sigma \to \tau$. This is calculated summing 1 for each atom and each '$\to$' in $\sigma \to \tau$.

Definition 2.1. *With each λ-term $M \in \Lambda^{st}$ we associate an element of the ordinal ω^2 by setting $|M| = (k, n)$ where k is the maximal length of a redex in M and n is the number of redexes of length k occurring in M.*

In order to associate a suitable reduction strategy $\to_s$ guaranteeing that $M \to_s N$ entails $|M| > |N|$ we have to study how the contraction of a redex can duplicate other redexes or create new redexes. Duplication of a redex R happens when contracting redexes of the form

$$(\lambda x^\sigma.M[x,x]^\tau)^{\sigma\to\tau}R^\sigma \to_\beta M[R,R]^\tau,$$

where $M[P,Q]$ is a notation to display subterm occurrences of M.

The duplication of R is not very dangerous, while the creation of new redexes might be more problematic: *a priori* new redexes of higher length might be created indefinitely. The main instrument to check that this is not possible is given by the following lemma.

Lemma 2.2 (Creation of redexes, Lévy (1978))**.** *Contraction of a β-redex can only create a new redex in one of the following ways:*

(i) $(\lambda x^{\sigma\to\tau}.M[xP])(\lambda y^\sigma.Q) \to_\beta M[(\lambda y^\sigma.Q)P]$;

(ii) $(\lambda x^\sigma.(\lambda y^\tau.M[x,y]))PQ \to_\beta (\lambda y^\tau.M[P,y])Q$;

(iii) $(\lambda x^{\sigma\to\tau}.x)(\lambda y^{\sigma}.P)Q \to_{\beta} (\lambda y^{\sigma}.P)Q$.

As proved by Lévy in his PhD thesis (§1.8.4, Lemme 3), the above lemma holds more generally for the untyped lambda calculus. See also Exercise 14.5.3 in Barendregt (1984).

Lemma 2.3. *Suppose $M \xrightarrow{R}_{\beta} N$, i.e. N is obtained from M by contracting R, and let R' be a created redex in N. Then $\ell(R) > \ell(R')$.*

Proof. Check that in each case of Lemma 2.2 the property holds. □

This lemma is not explicitly mentioned in Turing's proof, but it is stated that when reducing a redex of highest length, no other redex of highest length is created.

The reduction strategy $\mathcal{S}$ taken by Turing for proving the weak normalization property is as follows. If M is in β-normal form, then do nothing; otherwise $\mathcal{S}(M) = N$ by contracting the *rightmost redex* of maximal length in M.

Theorem 2.4 (Weak Normalization). *The simply typed lambda calculus is weakly normalizing, i.e., every $M \in \Lambda^{st}$ has a β-normal form found by $\mathcal{S}$.*

Proof. Contracting a redex R can only duplicate redexes R' to the right of R. Since the redex R chosen by $\mathcal{S}$ is the rightmost of maximal length, it only duplicates redexes R' such that $\ell(R') < \ell(R)$. By Lemma 2.3 also the new redexes created by the reduction are of smaller length. Therefore $\mathcal{S}(M) = N$ entails $|M| > |N|$. As ω^2 is well-founded, we are done. □

A recent discovery is that Gentzen already had a normalization proof for derivations in natural deduction, see von Plato (2008). This implies the normalization of typed lambda terms. However, the proof worked out for lambda calculus is more clear and understandable, thanks to the simple syntax of λ-terms.

Strong Normalization

Actually the simply typed lambda calculus enjoys strong normalization, which means that all β-reductions are terminating regardless of the strategy that is chosen. The classic proof of strong normalization by using the reducibility technique is due to Tait (1967), already obtained in 1963 and used by many authors. The proof of strong normalization by Tait does not use a complexity measure assigned to terms. In de Vrijer (1987) it is shown that it is possible to do this, assigning to a term M an ordinal $|M|$ (in fact a natural number), in such a way that $M \to_\beta N$ entails $|M| > |N|$, regardless what redex is reduced. It is an open problem whether such ordinals can be assigned in a natural and simple way.

3. Postscript

Lambda calculus was more often on Turing's mind. The logician Robin Gandy, who had been a student and associate of Turing, mentioned in 1986 at a conference for his retirement, that in the early 1950s Turing had told him ideas to implement lambda reduction using graphs. This is now commonly done when designing compilers for functional programming languages. Thereby Turing was not careful about the distinction between free and bound variables and Gandy could correct him. Then Turing said: "That remark is worth 10 pounds a week!", in those days enough for a decent living.

References

Barendregt, H. P., 1984. The lambda calculus, its syntax and semantics, 2nd Edition. No. 103 in Studies in Logic and the Foundations of Mathematics. North-Holland.

Barendregt, H. P., Manzonetto, G., Plasmeijer, M. J., 2012. The imperative and functional programming paradigm. In: Cooper, B., van Leeuwen, J. (Eds.), This volume. Elsevier, pp. xxx–xxx.

Endrullis, J., Hendriks, D., Klop, J. W., 2010. Modular construction of fixed point combinators and clocked Böhm trees. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom. IEEE Computer Society, pp. 111–119.

Gandy, R. O., 1980. An early proof of normalization by A. M. Turing. In: Seldin, J. P., Hindley, J. R. (Eds.), To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press Limited, pp. 453–455.

Lévy, J.-J., 1978. Réductions correctes et optimales dans le lambda-calcul. Ph.D. thesis, Université Paris 7.

von Plato, J., 2008. Gentzen's proof of normalization for natural deduction. The Bulletin of Symbolic Logic 14 (2), 240–257.

Tait, W. W., 1967. Intentional interpretation of functionals of finite type I. The Journal of Symbolic Logic 32 (2), 198–212.

Turing, A. M., 1937. The p-function in lambda-K-conversion. The Journal of Symbolic Logic 2 (4), 164.

de Vrijer, R. C., 1987. Exactly estimating functionals and strong normalization. Indagationes Mathematicae 49, 479–493.