MINIMISATION VS. RECURSION ON THE PARTIAL CONTINUOUS FUNCTIONALS

Ulrich Berger
Department of Computer Science
University of Wales Swansea

Abstract We study the relationship between partial continuous higher type functionals defined by minimisation on the one hand and recursion on the other hand. We prove that already at type level two minimisation is weaker than recursion.

1. Introduction

There are two well-known ways of extending the schemata for the primitive recursive functions such that all partial recursive functions are obtained. One is recursion the other is minimisation. In Kleene (1959a) both are extended to higher types: recursion via the schemata (S1-S9) and minimisation via μ-recursion. Kleene showed that at type two (S1-S9) and μ-recursion coincide, but already at type three the latter is weaker than the former. In Bergstra (1976) it is proved that also on the total continuous functionals (Kleene 1959b, Kreisel 1959) the schemata (S1-S9) are weaker than recursive continuity (having a recursive associate). These results were based on an interpretation of computations on *total* (continuous) functionals, whereas it is now common to interpret them on the *partial* continuous functionals. Normann showed that this makes a difference by proving that under the new interpretation (S1-S9) is as powerful as recursive continuity when restricted to the total continuous functional (Normann 2000). To be precise Normann works with the functional language PCF (Plotkin 1977), which however, as essentially shown in Platek (1966), is equivalent to (S1-S9) on the partial continuous functionals.

In this paper it is shown that the relationship between minimisation and recursion is affected by the choice of the underlying domain, too. We prove that already at type level two minimisation is weaker than recursion. From our result we deduce that at those types minimisation is not only denotationally but also operationally weaker than PCF.

P. Gärdenfors, J. Woleński and K. Kijania-Placek (eds.), In the Scope of Logic, Methodology and Philosophy of Science, Vol. I, 57-64.

After giving in section 2 the basic definitions we present in 3 as a starter a new short proof that every computable monotone functions of type level one is μ-recursive in the parallel OR, a result first proved in Trakhtenbrot (1976). Section 4 contains the main result already discussed above, and in section 5 we conclude by discussing some open problems arising at higher types.

2. Partial continuous functionals

The hierarchy of *partial continuous functionals* is a family of effective Scott-Ershov-domains (Scott 1982, Griffor, Lindström and Stoltenberg-Hansen 1993) defined by

$$D_\iota := \mathbf{N}_\perp, \quad D_o := \mathbf{B}_\perp, \quad D_{\rho\to\sigma} := D_\rho \to D_\sigma,$$

where $\mathbf{N}_\perp := \mathbf{N} \cup \{\perp\}$ and $\mathbf{B}_\perp := \mathbf{B} \cup \{\perp\}$ ($\mathbf{B} = \{\mathrm{tt}, \mathrm{ff}\}$) are the flat domains of partial integers and boolean values respectively. $D \to E$ denotes the domain of continuous functions from D to E, i.e. the exponential in the cartesian closed category of Scott-Ershov domains and continuous functions. We let $\mathcal{D}$ be the union of the D_ρ. By a *functional* (of type ρ) we mean an element of $\mathcal{D}$ (D_ρ). When writing $f\colon \rho$ we mean that f is a functional of type ρ. A functional of type ρ is called *computable* if it is a computable element of the effective Scott-Ershov domain D_ρ, i.e. its compact approximations are recursively enumerable.

We let τ range over the base types ι and o. A type $\rho_1 \to (\rho_2 \to \ldots \to (\rho_n \to \sigma)\ldots)$ will often be written $\rho_1 \to \rho_2 \to \ldots \to \rho_n \to \sigma$ or simply $\vec{\rho} \to \sigma$. An iterated application $f(x_1)\ldots(x_n)$ will also be written $fx_1\ldots x_n$ or $f(x_1,\ldots,x_n)$. For $\rho = \vec{\rho} \to \tau$ we define $\perp_\rho\colon \rho$ by $\perp_\rho(\vec{x}) := \perp$. The level of a type ρ is defined as usual by $\mathrm{lev}(\iota) = \mathrm{lev}(o) := 0$, $\mathrm{lev}(\rho \to \sigma) := \max(\mathrm{lev}(\rho)+1, \mathrm{lev}(\sigma))$. The level of a functional $f \in D_\rho$ is the level of ρ. A functional $g \in \mathcal{D}$ is *explicitly definable* from a set of functionals $\mathcal{F} \subseteq \mathcal{D}$ if g can be defined from elements of $\mathcal{F}$ by application and λ-abstraction.

In the following we will frequently omit type information as long as it can be recovered from the context.

A functional of type level $\leqslant 1$ will be called a *monotone function*. The following monotone functions will be used frequently (see also Plotkin (1977)). We let i, j, m, n, k range over natural numbers.

$\mathrm{if}_\tau\colon o \to \tau \to \tau \to \tau$	$\mathrm{if}_\tau(\mathrm{tt}, x, y) = x$, $\mathrm{if}_\tau(\mathrm{ff}, x, y) = y$, $\mathrm{if}_\tau(\perp, x, y) = \perp$
$\mathrm{Z}\colon \iota \to o$	$\mathrm{Z}(0) := \mathrm{tt}$, $\mathrm{Z}(n+1) := \mathrm{ff}$, $\mathrm{Z}(\perp) := \perp$
$(+1)\colon \iota \to \iota$	$(+1)(n) := n+1$, $(+1)(\perp) := \perp$
$(-1)\colon \iota \to \iota$	$(-1)(0) := 0$, $(-1)(n+1) := n$, $(-1)(\perp) := \perp$

The set $\{0, \mathrm{if}, \mathrm{Z}, (+1), (-1)\}$ is called the set of *basic functions*.

Recursive definitions in $\mathcal{D}$ are modelled by the *fixed point operators* $Y_\rho\colon (\rho \to \rho) \to \rho$,

$$Y(f) := \text{ the least fixed point of } f = \bigsqcup\{f^n(\bot_\rho) \mid n \in \mathbf{N}\}.$$

A functional is definable in the functional programming language PCF (Plotkin 1977) iff it is explicitly definable from the basic functions and the fixed point operators.

A functional is *μ-recursive* if it is explicitly definable from the basic functions, the *primitive recursor* $R\colon \iota \to (\iota \to \iota \to \iota) \to \iota \to \iota$,

$$R(x,h,0) := x, \quad R(x,h,n+1) := h(n,R(x,h,n)), \quad R(x,h,\bot) := \bot,$$

and the *minimisation functional* $\mu\colon (\iota \to o) \to \iota$,

$$\mu(f) := n \text{ if } f(n) = \mathrm{tt} \text{ and } f(i) = \mathrm{ff} \text{ for } i < n, \quad \mu(f) := \bot \text{ otherwise}$$

A functional g is *μ-recursive in a set of functionals* $\mathcal{F}$ if there are $f_1, \ldots, f_k \in \mathcal{F}$ and a μ-recursive functional h such that $g = h(f_1, \ldots, f_n)$. The notions '*PCF-definable in*' and '*computable in*' are defined similarly.

The following implications are well-known:

$$\mu\text{-recursive} \quad \Rightarrow \quad \text{PCF-definable} \quad \Rightarrow \quad \text{computable}$$

Of course these implications also hold when relativised to a set of functionals.

A *strict function* is a monotone function f such that $f(\vec{x}) = \bot$ whenever $x_i = \bot$ for some i. It follows from ordinary recursion theory that a strict function is computable iff it is μ-recursive.

3. Monotone functions

In Sazonov (1975) it is proved that a monotone function is PCF-computable iff it is an *effectively sequential function* (Vuillemin 1975). In Trakhtenbrot (1976) it is shown that the effectively sequential functions are precisely those definable from computable strict functions and if_τ by composition. Since computable strict functions are μ-recursive we have:

Theorem 1 (Sazonov, Trakhtenbrot) *A monotone function is PCF-definable iff it is μ-recursive.*

The monotone function *parallel or* $OR\colon o \to o \to o$ is defined by

$$OR(\mathrm{tt},x) = OR(x,\mathrm{tt}) = \mathrm{tt}, OR(\mathrm{ff},\mathrm{ff}) = \mathrm{ff}, OR(x,y) = \bot \text{ otherwise.}$$

Note that $OR(\mathrm{tt},\bot) = OR(\bot,\mathrm{tt}) = \mathrm{tt}$, but $OR(\bot,\bot) = \bot$. Hence OR is not sequential and therefore not PCF-definable (although computable, trivially).

http://www.springer.com/978-1-4020-0929-7

In the Scope of Logic, Methodology and Philosophy of Science
Volume One of the 11th International Congress of Logic, Methodology and Philosophy of Science, Cracow, August 1999
Gärdenfors, P.; Wolenski, J.; Kijania-Placek, K. (Eds.)
2003, XV, 384 p., Hardcover
ISBN: 978-1-4020-0929-7