A Combinatory Account of Internal Structure

Barry Jay and Thomas Given-Wilson
QCIS, School of Software
University of Technology, Sydney

Sapling, Sydney, 2nd October, 2009

Abstract

Traditional combinatory logic is able to represent all Turing computable functions on natural numbers, but there are effectively calculable functions on the combinators themselves that cannot be so represented, because they have direct access to the internal structure of their arguments. Some of this expressive power is captured by adding a factorisation combinator. It supports structural equality, and more generally, a large class of generic queries for updating of, and selecting from, arbitrary structures. The resulting combinatory logic is *structure complete* in the sense of being able to represent pattern-matching functions, as well as simple abstractions.
Full draft paper at `www-staff.it.uts.edu.au/~cbj/Publications/factorisation.pdf`

Pattern Calculus

Pattern Calculus
Computing with Functions and Structures

is out! `http://www.springer.com/978-3-540-89184-0`.

Also, **bondi** 2.0 is available from `bondi.it.uts.edu.au`.

Question: What is the relationship between pattern calculus and λ-calculus?
Answer: Difficult! Dynamic pattern calculus supports *matchable symbols* as well as *variable symbols*.

Idea: Avoid variables by using combinatory logic.
Avoid pattern matching by examining internal structure.

SK-logic

A combinatory logic is given by some *atoms* A from which are built *combinators*

$$M, N ::= A \mid MN$$

which support reduction rules (or equations). A popular, and important, example is SK-logic given by

$$A ::= S \mid K$$

and *reduction rules*

$$\begin{array}{rcl} SMNX & \longrightarrow & MX(NX) \\ KXY & \longrightarrow & X\,. \end{array}$$

The combinator I satisfying $IX \longrightarrow X$ can be represented by SKK (or any SKY). SK-logic supports λ-abstraction.

SF-logic

Now consider internal structure, by *factorisation*.

$$A ::= S \mid F$$

The *matchable forms* are $S \mid SM \mid SMN \mid F \mid FM \mid FMN$. A *compound* is a matchable form that is also an application. Its reduction rules are

$$\begin{array}{rcll} SMNX & \longrightarrow & MX(NX) & \\ FAMN & \longrightarrow & M & \text{for any atom } A \\ F(PQ)MN & \longrightarrow & NPQ & \text{if } PQ \text{ is a compound.} \end{array}$$

Now matchable forms are exactly the head-normal forms.
Also K is represented by FF.
Factorisation can be used to define equality of normal forms, and other path polymorphic functions (generic queries).

F is not definable from S and K

Theorem: F is not definable from S and K.
Proof Sketch: The translation of SKY to λ-calculus eliminates Y but F can recover it.

SF-logic is structure complete

A combinatory logic is *structure complete* if it supports static pattern-matching functions (as described in the paper).

Theorem: SF-logic is structure complete.
Proof Sketch: Define matching by induction on the pattern.

Theorem: SK-logic is *not* structure complete.
Proof: Consider factorisation acting upon SK-logic

$$\begin{array}{rcl} \mathcal{F}(A, M, N) & = & M \\ \mathcal{F}(PQ, M, N) & = & NPQ \text{ if } PQ \text{ is head normal} \end{array}$$

$\mathcal{F}$ would be represented by F but this is not definable here.

Conclusions

Theorem: F is not definable from S and K.
Theorem: SK-logic is *not* structure complete.
Theorem: SF-logic is structure complete.

Although SK-logic is combinatorially complete (supports λ-abstraction), it is not complete, as one may add more combinators to add expressive power.

F adds the power of pattern matching, which about covers what one may do with normal forms. However, there may be other combinators for, say, identifying fixpoints.

SF-calculus allows a single combinator to be both a function that can be evaluated, and a data structure, that can be examined, making possible dynamic program transformation.