The Expressive Power of Higher-order Types or, Life without CONS

Neil D. Jones

March 5, 1999

Abstract

We study the expressive power of functional programming language features by characterizing decision problems solvable with various combinations of *operations on data:* constructors and destructors; the order of *data values:* 0, 1, or higher; and program *control structures:* general recursion, tail recursion, primitive recursion.

Many combinations are Turing-complete, so such programs compute everything computable: all the partial recursive functions. A classic Turing-incomplete language is got by restricting data to order 0 and control to "fold." Such programs compute the *primitive recursive* functions.

The table shows the effect of higher-order types on the computing power of programs of type `[Bool]->Bool`. Each entry is a complexity class: the collection of decision problems solvable by programs restricted by row and column indices. RO stands for "read-only," i.e., constructor-free programs, and RW stands for "read-write."

Program class	**Data order 0**	**Order 1**	**Order 2**	**Order 3**	...
RW = read-write	REC. ENUM.	REC. ENUM.	REC. ENUM.	REC. ENUM.	...
RWPR, fold as control (*primitive recursion*)	PRIM.REC.	$\text{PRIM}^1\text{REC.}$	$\text{PRIM}^2\text{REC.}$	$\text{PRIM}^3\text{REC.}$	...
RO = read-only	PTIME	EXPTIME	EXP^2TIME	EXP^3TIME	...
RO tail recursive	LOGSPACE	PSPACE	EXPSPACE	EXP^2SPACE	...
ROPR, fold as control	LOGSPACE	PTIME	PSPACE	EXPTIME	...