© 2013 FIZ Karlsruhe

io-port 05664262

Sperber, Michael; Dybvig, R.Kent; Flatt, Matthew; van Straaten, Anton; Findler, Robby; Matthews, Jacob

Revised^6 report on the algorithmic language scheme.

J. Funct. Program. 19, Suppl., 1-301 (2009).

Summary: Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary. Scheme demonstrates that a very small number of rules for forming expressions, with no restrictions on how they are composed, suffice to form a practical and efficient programming language that is flexible enough to support most of the major programming paradigms in use today. Scheme was one of the first programming languages to incorporate first-class procedures as in the lambda calculus, thereby proving the usefulness of static scope rules and block structure in a dynamically typed language. Scheme was the first major dialect of Lisp to distinguish procedures from lambda expressions and symbols, to use a single lexical environment for all variables, and to evaluate the operator position of a procedure call in the same way as an operand position. By relying entirely on procedure calls to express iteration, Scheme emphasized the fact that tail-recursive procedure calls are essentially gotos that pass arguments. Scheme was the first widely used programming language to embrace first-class escape procedures, from which all previously known sequential control structures can be synthesized. A subsequent version of Scheme introduced the concept of exact and inexact number objects, an extension of Common Lisp's generic arithmetic. More recently, Scheme became the first programming language to support hygienic macros, which permit the syntax of a block-structured language to be extended in a consistent and reliable manner.

Keywords: lambda calculus; Common Lisp's generic arithmetic
doi:10.1017/S0956796809990074