An Overview of the ECL Programming System

Ben Wegbreit

Center for Research in Computing Technology
Harvard University

The ECL programming system has been designed as a tool for tackling 'difficult' programming projects. That is projects on which existing languages could be used only with considerable waste in machine or programmer time.

Such projects include much of the frontier of computer technology; whenever several application areas are conjoined and whenever solution of a problem requires linguistic development -- in algorithmic notation or information structures. Examples range from the management of large scale distributed data bases to applied artificial intelligence.

Specifically, projects of this nature are systems characterized by two requirements:

(1) Considerable experimentation is required to develop the system; that is, the design and development of the system must go hand in hand. Typically, this occurs when problems are so complex that significant computer assistance and experimentation are needed in system design.

(2) When a complete system is ultimately designed and programmed, it must be possible to take the working programs and produce a highly efficient product - both in machine time and space - with no change to the basic algorithms or their representation.

The ECL system has been designed as a vehicle for such undertakings. At the present time an experimental version of the system is operational - a version which only partially meets the above requirements. Additional system development is underway and will continue for some time. This paper outlines the goals of this work and the ECL system as planned.

The ECL programming system consists of a programming language, called ELl, and a system built around that language to provide a complete environment for the human-oriented use of the language. The system allows on-line conversational construction, testing, and running of programs. It includes an interpreter, a fully compatible compiler, and an editor -- all callable at run-time, on programs constructible at run-time either by the programmer or as the result of computation.

ELl is an extensible language. Thus it provides a number of facilities for defining extensions so that the programmer can readily shape the language to the problem at hand, and progressively reshape the language as his understanding of the problem and its solution improves. Like the familiar notions of subroutine and macro definition, these extension facilities allow one to abstract siginificant aspects of a complex algorithm. Such functional

abstraction serves both as a representational aid and as a handle on the production of an efficient product.

Specifically, the language provides facilities for extension of four axes: syntax, data types, operations, and control.

(1) Syntactic extension allows the specification of new linguistic forms and their meaning in terms of existing forms.

(2) Data type extension allows the programmer to define new data types and new information structures whenever needed to model the problem at hand. EL1 is significant in this regard in that considerable attention has been given to the efficient representation of programmer-defined data types. There is a special compiler for data-type definitions which computes space-efficient packing of structures into machine words, and generates machine code for rapid handling of objects and their components.

(3) Operator extension allows the programmer to define new operations on new data types and to extend existing operations to cover new types. There are two key points here:

   (a) Operators and procedures are not restricted to act on built-in types in the language but can, and in general will, take arguments whose mode is programmer-defined.

   (b) Declarations can be made to allow the compiler to perform type-checking and type-conversion code generation. Hence, it is possible to write programs operating on extended data types whose execution at run-time is comparable to that for using built-in modes.

(4) Control extension allows the creation, deletion, and coordination of independent asynchronous processes, called paths in ECL. The extension mechanisms are sufficiently flexible that co-routines, the P and V operations of Dijkstra, multiple parallel returns, and path scheduling are all definable in the system as proper extensions. Hence, it is straightforward to program almost all known control structures as well as an unknown variety of others.

In addition to these definition mechanisms provided by the language, the ECL programming system provides a number of other handles which the programmer can use to extend and tailor the environment in which he operates. Many of the system's facilities are written in the language and hence open to argument , modification, and replacement by the programmer. These include the compiler, the editor, and most of the input/output and file system.

Extensibility alone is, however, not sufficient; its counterpart -- contractibility -- is also required. That is, having produced running prototype programs, it must be possible to subject these the programs to a sequence of contractions -- commitments with respect to subsequent nonvariation so as to obtain a final system optimal for the project requirements.

The ECL programming system and the EL1 programming language have been designed to allow this. Programs can be run either by an interpreter or a fully compatible compiler. Compiled and interpreted functions can call each other in either direction. Compilation can itself be progressively refined. The programmer is free to supply as much declarative information as he wishes (or knows) at a given time and the compiler will do the best it can with the information given. Successive recompilation with additional information will produce progressively better code. Such contractions include: evaluation of arbitrary expressions at compile-time, fixing the values of procedures and operators, fixing the lengths of arrays, and fixing the data types of both local and free variables. For example, one can specify that the data type of a variable fall into any of the following categories: 1) it is completely dynamic at run-time, 2) it is restricted to a specific set of fixed modes, 3) it is a fixed mode but its length is not fixed, 4) its mode and size are both fixed. The programmer can choose, for each variable, in which category it is to fall, and he can choose again -- with minimal programming effort -- as the project progresses.

In summary, the intended application of the ECL programming system is the programming project which would otherwise be prohibitively uneconomical. To this end, the ECL system has been designed to allow flexible programmer-oriented program construction and testing coupled with facilities for subsequent optimizing contractions which produce an efficient final product.

Companion papers by C. Prenner, G. Holloway, and B. Brosgol describe specific aspects of the ECL system and its implementation: control extensions, data type extensions, and compilation.

BIBLIOGRAPHY

Wegbreit, B., "The ECL Programming System," Proc. FJCC, Vol. 39, 1971, pp. 253-262.

Wegbreit, B., "The Treatment of Data Types in EL1," Center for Research in Computing Technology, Harvard University, Cambridge, Mass., 1971.

The design and implementation of the ECL programming system was supported in part by the U.S. Air Force, Electronics System Division under contract F19628-71-C-0173 and by the Advanced Research Projects Agency under contract F19628-68-C-0379.