Escaping from intermediate expression swell: a continuing saga

J.A. Padget,
School of Mathematics, University of Bath,

Abstract
The notion of a closed continuation is introduced, and a functional dialect of LISP is presented, using function call and return based on this concept, by extension coroutines are shown to be merely a more general form of function. Potential applications for coroutines in algebraic simplification and expression evaluation are suggested, and a specific example of their use is given in a novel attack on the phenomenon of intermediate expression swell in polynomial multiplication.

Introduction
In essence there are two groups in the field of algebraic manipulation, those who use LISP and the rest. Whilst remaining impartial to their respective merits and demerits, it seems that LISP, is not so much being used as abused, because only the equivalents of the data and control constructs of less high level languages are generally employed. Even worse, some LISP implementations (Standard LISP [Marti et al]) sidestep the classical 'problem' of the FUNARG - in effect ALGOL semantics with LISP syntax.

LISP is popular for algebra systems through features such as property lists, identity of algebraic variables with program variables, ease of substitution and of course recursion. The more important and advanced features such as the passing of functions and environments as arguments or returning them as values of expressions (downward and upward FUNARGS respectively), continuations, and closures are ignored - all these can be used to provide advanced control regimes to the good of algebra.

It is the intention of this paper to show how coroutines can solve some algebra problems in a simpler, clearer style when the solution to a problem is logically parallel, yet in the restricted LISP environment it was translated to a sequential program. To reflect this logical parallelism, control structures are required to go beyond a single stack-based system. In consort with recursion, coroutines are an extremely powerful and flexible medium for specifying algorithms: recursion gives the dynamic allocation and coroutines provide context switching.

In Functional Basis the proposed scheme for function invocation is described, then developed in Coroutine Control to give the necessary primitives for use in Applications of Coroutines. In conclusion there is the Summary and an outline of Further Work. The appendix is a partial denotational semantics of a functional dialect of LISP.
Functional Basis

The proposed basic control element is the closed continuation, a practical manifestation of the expression continuation [Strachey&Wadsworth] in a dynamically scoped language. The expression continuation was introduced into denotational semantics [Scott&Strachey] as a means of handling full jumps as proposed in PAL [Evans]. The continuation of an expression is the 'rest of the program'; that is where an expression will pass its result. In practice a function is needed to perform an encapsulation of the current environment; call it BCNU - Binding Context Node Utility.

Thus an alternative method of function invocation can be based on the theme of jump and continue rather than call and return, whereby a function is evaluated by first saving the current environment, second setting up the arguments and then jumping to the function body; upon termination the continuation is evaluated. The continuation becomes a first-class citizen and can be used for elegant and consistent description of control flow. For example: a loop is a tail-recursion, which is a recursion where the initial continuation is repeatedly passed forward; hence the recursion is unfolded. This is similar to the concept of ACTORS [Hewitt] except that they only exist in isolated lexical environments. A generalisation of this model in which continuations are created and passed around arbitrarily leads to the coroutine.

General jumps are the key to directing control flow in the applicative implementation of coroutines. This is done by CATCH and THROW, their behaviour being a parallel to ERRORSET and ERROR, but intended for controlling the direction of evaluation rather than for handling exceptions. For the interested reader a denotational semantics for a purely applicative subset of LISP is given in the appendix. These semantics are not an afterthought, they provided the initial basis for the discussion and investigation of coroutines. The point of using semantic descriptions is that in general language design is far too baroque; syntax has been given precedence over semantics, this is an attempt to follow a more formal, rigorous and mathematical approach, and so stress the important features.

Coroutine control

A coroutine is a function with more than one entry and more than one exit point and which preserves the environment in which it was suspended. When restarted, execution will continue from the point of suspension and in the environment as it existed at the time of suspension. There are three places to which it would be useful to pass control:-

(i) back to the caller (parent)
(ii) down to another function/coroutine (implicit)
(iii) over to another coroutine (sibling)