THE BARREL CONCEPT: A STUDY IN LANGUAGE SYSTEM DEVELOPMENT

Cindy Eades
GTS Computer Systems Inc.
Birmingham, Alabama
University of Alabama in Birmingham
Kevin D. Reilly
University of Alabama in Birmingham
John H. Barrett
University of Alabama in Birmingham
Charles Minderhout
U.S. Steel, Fairfield Works
University of Alabama in Birmingham

We describe a variation in theme on abstract machine implementation through general purpose macro processing. Using data flow diagrams we show how the central focus of concern can be shifted from the output focus of conventional macro processing to an user-oriented focus, on a system developed upon an optimized and extended version of the Stage2 processor of W. Waite and co-workers.

The approach has potential theoretical interest in its: being a modern expression of widely accepted older ideas and implementations, applications which incorporate synergisms in language concepts (string and list processing, tables), possible opening to logic programming.

Data flow descriptions are used to illustrate top-level and selected lower level computation activities, e.g., combination evaluation. Usage of the array of capabilities presented by Barrel are outlined: portability, prototyping in a multiple-machine context, "permanent" (compiled) codes for network operations.

I. Introduction

Abstract machines can be implemented in a variety of ways (Brown, 1974). Among them are techniques based on (general purpose) macro processing. Figure 1 provides a standard view of such an implementation, using data flow diagrams (Gane and Sarson, 1979).

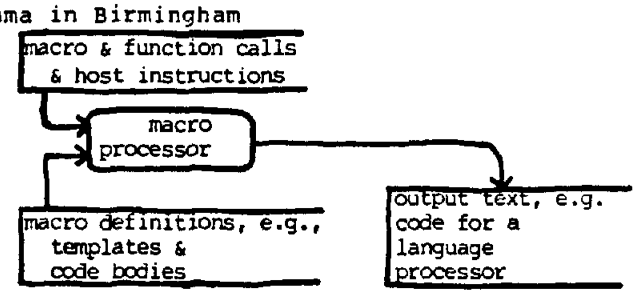

Figure 1: Data flow diagram for the top-level description of conventional macro processing (Scratch file(s) not included).

II. Top-Level Dataflow for Barrel

In some cases the processor can provide comprehensive facilities of its own, this being so more often for general purpose macro processors than for those limited or restricted to a particular host language. In such cases it is often possible to adapt (and extend) the processor so that it has the additional power needed, e.g., to serve as a facility for prototyping and development studies.

Such a prospect lies at the heart of the Barrel concept: the Stage2 general purpose macro processor (Waite, 1973) has been adapted and extended to forge a flexible tool for studies in a variety of areas: string and list processing, table-based processing methods, systems for support of analysis and design, and possibly logic programming.

The conceptual basis for this processing approach is of interest in its own right, and is displayed in Figure 2 (a figure which resembles Figure 1 in many respects, but which contains some changes in orientation and philosophy).

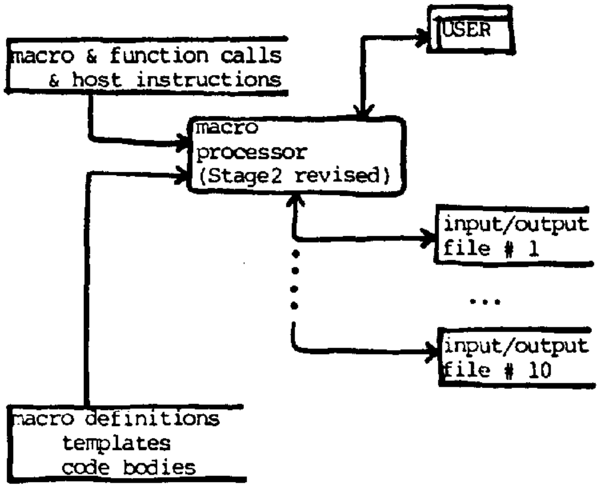

Figure 2: Data flow diagram for the top-level (simplified) description of the modified approach in which the augmented facilities of the macro processor (Stage2) are used interactively to forge a computing system for prototyping and development studies.

Figure 2 has been kept at a simple level to dramatize the change of focus that has occurred in our adaptation. What is only scarcely hinted in the figure is the prodigious effort of one of the authors to optimize the processor.

Special attention should be called to the extended file processing capabilities. These lie at the heart of interpretation and toward compilation. These are also necessary for table-processing applications. Other potentials also exist for them.

Changes in other parts of the system are under investigation particularly with respect to portions of the memory scheme to conform to certain theoretical models of memory organization and to make more efficient various kinds of computation. Hitherto, we have tried to avoid premature changes until a number of feasibility questions have been answered.

III. Categories of Use

In passing we mention that the system is being used for studies in four categories. First and minimally, the system can be used in straight-forward (perhaps student-exercise level) computations involving the interactive and batch facilities presented in a definitional file entitled BMAC. At this level the system resembles Basic, in the style of interacting with the system, but differs from Basic in having only structured programming control statements which are modeled after Pascal and Ada.

At a second level, the system provides string processing facilities modeled after those of the new programming language, Icon, a product resembling Snobol, but with a different set of primitive constructions and a new approach to pattern matching. The definitional files for this type of work are entitled BICON (for Barrel Icon), and presume the use of BMAC in most cases. They are, however, independent of the other definitional files to be described next. BICON is presently still in the early state of development.

BLISP is the third component, and, as already mentioned, can run independent or in conjunction with BICON. BLISP is an evolving subset of LISP, e.g., it does not have a "go" (within "PROG") though the control structures of BMAC could be used to implement structured iteration.

The fourth and final part of the definitional structures is BTPS constructions for table-processing. By April 1982 BTPS should allow some variety of table input, e.g., decision table, condition policy maps and action policy maps (Montalbano, 1974). BTPS, as well as each of the four areas of computation, is described in somewhat more detail in the companion paper.

IV. Combination Evaluation as an Example Development.

In this section we outline some of the considerations involved in development of an important facet of the BLISP part of the system: evaluation of combinations. As with every (significant) modification and/or extension of the system, we use a design and documentation approach in which we view the changes as a "scientific experiment". We assume a four-phase operation of : Purpose(1), Methods(2), Results(3) and Discussion(4). The next comments illustrate an abbreviated example of this approach.

PURPOSE:

This is "another" in the series of extensions to Barrel, specifically within BLISP. We seek, as usual, an adequate test of our new constructions. Our specific goal is to be able to process examples like: (cons (cons (cons y y) y) y), (car (cdr (crd x))) or (caddr x), (cons (car x) (cdr x)) and (cons (car (cdr x)) (car (cdr y))). These examples are chosen because they have characteristics which respectively are "heavy in the car (front-end), "heavy" in the cdr (back-end) and balanced between the car and the cdr.

Note that we are not trying to handle DEFINE in this study.

Our solution should be such that both interpretation and compilations are facilitated. We quote from Burge (1975):

> For combinations, the compiler is no more efficient than the interpreter; the same steps are merely carried out in a different order. The compiler version has been introduced to prepare the way for a more efficient method of evaluating expressions that contain lambda expressions. In this case, the body of a lambda expression may have to be evaluated more than once during the evaluation of an expression containing it; whereas it need only be compiled once.

METHODS:

Our approach (similar to that of Burge) can be overviewed in two steps as illustrated in Figure 3.

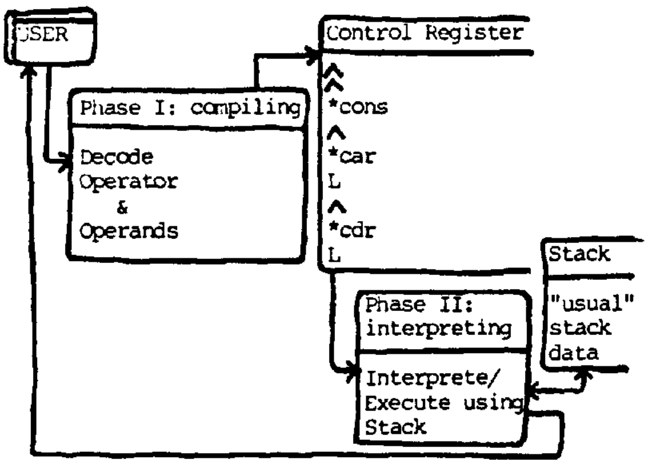

Figure 3: Dataflow description of the two-phase calculation for evaluation of the combination (cons (car L) (cdr L))

We paraphrase part of Burge's text to illustrate Phase I's decode ("compile") action on the User's input, resulting in the establishment of a control register and Phase II's interpreting/executing of the Control Register using a stack:

Phase I: Compiling

(p (m (p a b) c) (f a c))	original combination
(f x y) = $x^2 + y^2$	formula for the function f
3,1,f,A,A,3,2,1,p,A,A,m,A,A,p,A,A	Control Register

Phase II: Interpreting/Executing

Entries are taken from the Control Register and given to the Stack (S) until an A (or in our solution ^) is encountered. A check is made to see if another A (^) is encountered to distinguish binary (2 are found) or unary (only 1 is found) operation. The Stack (S) is now evaluated after which the process begins again.

The next section (Results) illustrates our rendition of our scheme within our Stage2 context.

RESULTS:

An attempt is now made to discuss the two phases in detail through a simple example.

Phase I: Decoding ("compiling")
The combination, e.g., (cons (cdr x) y), is first evaluated, which creates the Control Register (C): * denotes an internal function):

```
^
^
*cons
^
*cdr
x
y
___
(C)
```

Phase II: Executing
The Control Register (C) is evaluated (bottom up) and places the result in a Stack (S). The Stack (S) evolves as follows: (val denotes the value of):

(S)	(S)	(S)	(S)
*cdr val x val y	val(*cdr x) val y	*cons val(*cdr x) val y	val(*cons (cdr x) y) (returns value to the USER next)

The Stage2 code for these manipulations is compact (see Figure 4 for Phase I code for a slightly simplified case.)

The code has been analyzed according to its two phases. We are satisfied with Phase I. Phase II, however, runs somewhat slower than we would like (see Discussion for our future plans).

```
 (# #):
*(#10 #20)$
EVAL*$  GOOD POINT FOR EXAMINING CTRL REG.
$
*(# #):
 IF '#10 EQ 'CAR SKIP 4$
 IF '#10 EQ 'CDR SKIP 3$
 IF '#10 EQ 'ATOM SKIP 2$
 IF '#10 EQ 'NULL SKIP 1$
 (INSQHI '^)$
 (INSQHI '^)$
 (INSQHI '*#10)$
(#20#97 $
 (SETQ %SPU '#90)$
 %SPU1 := SECT(%SPU,1,1)$
 IF %SPU1 EQ LPAR SKIP 2$    RECURSE ?
 (INSQHI %SPU)$
SKIP 1$
*#90$
#F8$
$
```

Figure 4: Illustration of what it might be

like to (recursively) code the compiler phase in a case simplified to the basic LISP operations. The pattern base of the method, the use of INSQHI are among features of note.

DISCUSSION

The (relatively) successful results of this "experiment" are very important for future work on the BLISP component of Barrel, specifically as we progress to an evaluation facility comparable to that demanded by current (typical) functional programming.

The speed of evaluation, as remarked above, is only partially satisfactory. Fortunately, more than one possible cause can be hypothesized, and solutions to a couple of these are under consideration. A first hypothesis is that we have failed to exploit the basic pattern matching strength of Stage2. The remedy in this case is not trivial, but it is not difficult either. The second hypothesis is that "special" mechanisms may be needed to facilitate these schemes. The remedy in this case would be less painless, and should (and would) not be undertaken without its being needed for other reasons as well. We expect, however, to be probing our underlying code somewhat more in future work than we have in the past.

V. Concluding Remarks.

We have tried in this paper to illustrate the conceptual basis of the Barrel system and its categories of use, and to provide a (single) concrete example of a reasonably formalized development approach to achieve a discrete component of the software, i.e., combination evaluation. Future possibilities include continuation and extension of this approach to complement work in areas such as logic programming, pattern-directed, and table-driven processing, potentially in the network context.

REFERENCES

1. BROWN, P.J., Macro Processors and Techniques for Portable Software. N.Y.: Wiley-Interscience, 1974.

2. GANE, C. and T. Sarson, Structured Systems Analysis: Tools and Techniques. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

3. WAITE, W.M., Implementing Software for Non-Numerical Applications. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

4. MONTALBANO, M., Decision Tables. Chicago: SRA, 1974.

5. BURGE, W., Recursive Programming Techniques. Reading, Mass.: Addison-Wesley, 1975.