Can Forth liberate programming from the von Neuman style?

Language creation in Forth and Backus style functional programming

Key lessons

- Principles of array/functional programming
- Creating a “language” in Forth
- Demonstration: Chinese Multiplication Table

John Backus Achievements

- Invented FORTRAN
- Invented Backus Normal Form (BNF) for describing syntax
- Helped popularise functional programming
  - 1977 Turing Award
  - Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

5.1 A von Neumann Program for Inner Product

```
c := 0
for i := 1 step 1 until n do
  c := c + a[i]×b[i]
```

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

5.2 A Functional Program for Inner Product

Def Innerproduct
≡ (Insert +)∘(ApplyToAll ×)∘Transpose

Or, in abbreviated form:

Def IP ≡ (/+)∘(α×)∘Trans.

Language Creation in Forth

5.2 A Functional Program for Inner Product

Def Innerproduct
≡ (Insert +)∘(ApplyToAll ×)∘Transpose

Or, in abbreviated form:

Def IP ≡ (/+)∘(α×)∘Trans.

Forth is the reverse of Backus' notation.

\ *2 is multiplication for a two element list
\ e.g. <| 2 3 |> 2* == 6
: IP trans [‘] *2 all [‘] + ins ;

Just need to define the words used in the paper

Bump allocation can be used for simple examples

Internals

```
0 value pmem                                    \ pointer to free memory cell
0 value vlen                                    \ REGISTER vector length
0 value fun                                     \ REGISTER function
0 value lv                                      \ REGISTER (temp) last depth
0 value pvec                                    \ REGISTER pointer to vector
create mem MEM_SIZE allot                       \ create free memory pool
: reset mem MEM_SIZE + to pmem ;                \ deallocate all memory
: len @ ;                                       \ access vector length (stored in first cell of vector)
: pvec@+ pvec cell+ dup to pvec @ ;             \ auto increment and fetch pvec (C.H. Moore style)
: new pmem mem > 0= abort" out of mem" pmem swap cells - dup to pmem ;  \ memory allocation
: vec dup >r 0 do 1 new ! loop 1 new r> over ! ;                        \ create vector of N on stack
                                                \  Friendly vector syntax: <| x y z |>
: <| lv depth to lv  ;                          \  Save reg val on stack, record depth
: |> depth lv - vec swap to lv ;                \  Check change in depth and create vector, restore reg val
```

Example (live demo)

- Chinese multiplication table from 1 to 9

1. Produce all multipliers and multiplicands
2. Produce all products
3. Combine multipliers, multiplicands and products
4. Filter excess

- <| 1 1 1 |>
- <| 1 2 2 |> <| 2 2 4 |>
- <| 1 3 3 |> <| 2 3 6 |> <| 3 3 9 |>
- <| 1 4 4 |> <| 2 4 8 |> <| 3 4 12 |><| 4 4 16 |>
- <| 1 5 5 |> <| 2 5 10 |> <| 3 5 15 |><| 4 5 20 |><| 5 5 25 |>
- <| 1 6 6 |> <| 2 6 12 |> <| 3 6 18 |><| 4 6 24 |><| 5 6 30 |><| 6 6 36 |>
- <| 1 7 7 |> <| 2 7 14 |> <| 3 7 21 |><| 4 7 28 |><| 5 7 35 |><| 6 7 42 |><| 7 7 49 |>
- <| 1 8 8 |> <| 2 8 16 |> <| 3 8 24 |><| 4 8 32 |><| 5 8 40 |><| 6 8 48 |><| 7 8 56 |><| 8 8 64 |>
- <| 1 9 9 |> <| 2 9 18 |> <| 3 9 27 |><| 4 9 36 |><| 5 9 45 |><| 6 9 54 |><| 7 9 63 |><| 8 9 72 |><| 9 9 81 |>

What did I learn?

- Perfect Python interpreter: more than year and unusable
- Forth mini language: 2 days and usable
- Forth is excellent for exploring new languages and ideas. It is good for prototyping
- Function level programming works well for parallelism