Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England
t: +44 (0)23 8063 1441
e: sfp@mpeforth.com
w: www.mpeforth.com

Abstract

The VFX Forthv5.1 kernel incorporates dual-behaviour words and recognisers. This paper discusses our experience over the last year with these changes. Dual-behaviour words are a standards-compliant solution to needing words that have separate interpretation and compilation behaviour. Previous papers called these words NDCS words (non-default compilation semantics). Recognisers are a fashionable solution to providing a user-extensible text interpreter. Our experience converting two OOP packages to use recognisers is discussed.

Introduction

VFX Forth v5.1 was the first VFX version to provide dual-behaviour words and recognisers. An application of 1.34 million lines of Forth source code was converted to VFX 5.1 in 3.5 days and ran first time.

We adopted dual-behaviour words because they fix the problems previously solved by state-smart words. As implemented in VFX, dual-behaviour words are standard-compliant. The vast majority of the application conversion involved rewriting state-smart string-defining words. The application is a commercial one, and uses a large number of string types.

We adopted recognisers because they are fashionable and have one technical possibility that is important in large applications. It is currently impossible to persuade Forth programmers to use just one OOP package. Thus, if we are to reuse library code, we must learn how to manage multiple OOP packages. Recognisers provide a partial solution to this problem, but considerable attention to wordlist and naming is also required.

Dual-behaviour words

The classical Forth interpreter loop below is replaced by a new loop whose essential change is to distinguish between words with dual behaviour, defined in the ANS and Forth-2012 standards as “Non-Default Compilation Semantics” (NDCS for short). The term NDCS was liked by nobody, and words with this behaviour are now referred to as dual-behaviour words.

Traditional FIG-Forth interpreter

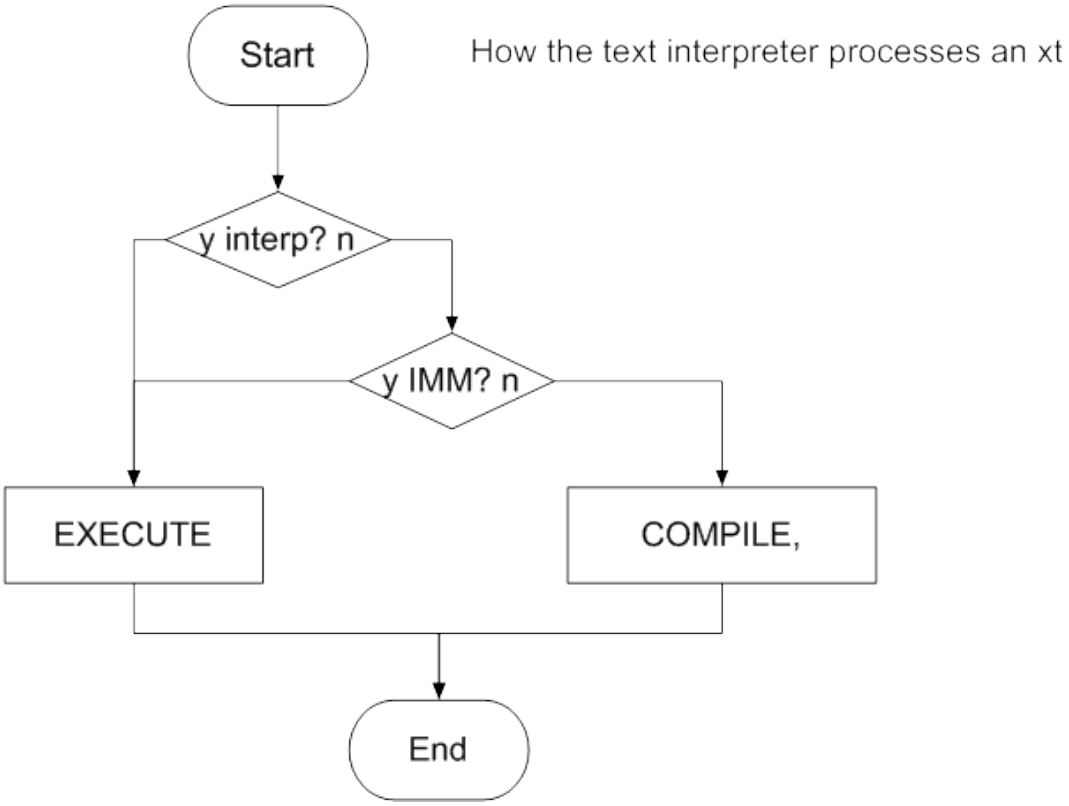
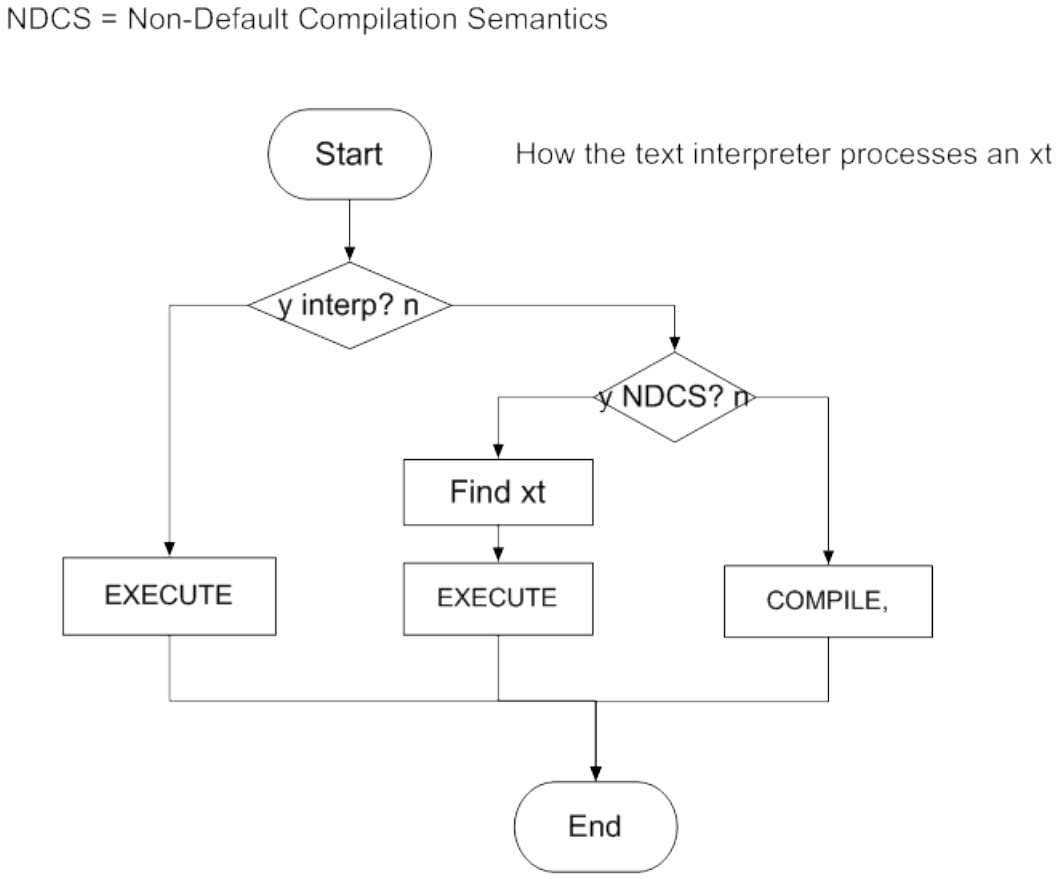


Illustration 1: Classical Forth interpreter loop

```
: process-xt    \ i*x xt -- j*x
  state @ 0 = if
    execute
  else
    dup immediate?
    if  execute  else  compile,  then
  then
;
```

The classical Forth interpreter loop has been used to describe the operation of Forth for over three decades now. It has been a useful model for many people. People regularly claim that they need to write a custom interpreter and that not all Forth systems permit this in a portable manner. We will see that a minor change to the loop and its associated structures brings it in line with Forth 2012 and expands the interpreter's facilities to take advantage of the Forth 2012 description of Forth words' action or behaviour or semantics. What we now call DUAL words are words such as **`IF`** that have separate interpretation and compilation behaviour, referred to in the standard as "non-default compilation semantics".

Dual-behaviour loop




```
: process-xt    \ i*x xt -- j*x
  state @ 0 = if
    execute
  else
    dup immediate? if
      execute
    else
      dual?
      if  dual,  else  compile,  then
  then
;
```

The picture illustrates a Forth interpreter/compiler loop that has been modified to cope with separated interpretation and compilation actions.

We also need a small number of new words that enable the loop to be constructed portably:
`IMMEDIATE?  Xt -- flag` ; return true if the word is immediate
`DUAL?  Xt -- flag` ; return true if the word has non-default compilation semantics
`DUAL,  i*x xt -- j*x` ; like **`COMPILE,`** but may parse.

In order to finish up, we need to understand what the word labelled DUAL`,` actually does. It finds the word that performs the non-default compilation semantics and then **`EXECUTE`**s it. The next picture shows the loop using the definition of **`IMMEDIATE`** words as having the same interpretation and compilation semantics.

The significant change is the introduction of a dictionary header flag, DUAL, which indicates that a word has non-default compilation semantics.

Since IMMEDIATE words are dual-behaviour words by definition, the interpreter processing of an xt can in theory be reduced to the one below

```
: process-xt    \ i*x xt -- j*x
  state @ 0 = if
    execute
  else
    dup dual?
    if  dual,  else  compile,  then
  then
;
```

The immediate flag has disappeared because all immediate words have non-default compilation semantics. They are immediate if the DUAL xt is the same as the for interpretation xt. The definition of immediate is more complicated in standards-speak, but comes to the same thing. An alternative implementation strategy may be to keep a separate immediate flag, but we should not hide the basic idea that immediate words have non-default compilation semantics.

The conventional immediate flag in a word's header becomes the DUAL flag, set for all words that have non-default compilation semantics. Comparison of the interpretation xt and the DUAL xt gives us a basis for the word **`IMMEDIATE?`** The word **`DUAL,`** just hides the system-specific action of obtaining the DUAL action from an xt.

Here's a potential way of building DUAL words. They illustrate a conventional **`IF`** … **`THEN`** pair. The word **`DUAL:`** modifies the previous word to have the following non-default compilation semantics – it defines a nameless word and sets system-specific flags and data.

```
: IF      \ C: -- orig ; Run: x --
\ This is the traditional interpretation behaviour
  NoInterp  ;
dual: ( -- orig )  s_?br>,  ;     \ conditional forward branch

: THEN     \ C: orig -- ; Run: --
\ This is the traditional interpretation behaviour
  NoInterp  ;
dual: ( orig -- )  s_res_br>,  ;  \ resolve forward branch
```

To produce an interpreted version, the interpretation behaviour is simply replaced by the new version. The next example shows how a contentious notation such as **`S"`** and friends becomes non-contentious.

```
: S"            \ Comp: "ccc<quote>" -- ; Run: -- c-addr u
\ Describe a string. Text is taken up to the next double-quote
\ character. The address and length of the string are
\ returned.
  [char] " parse >syspad
;
dual: ( -- ) postpone (s") ",  ;
```

Dual-behaviour words are discussed in more detail in my EuroForth 2018 paper "Implementing DUAL words"

Experience

While building the VFX kernel and system, we had few problems because we knew what we were looking for. When recompiling the CCS application, the majority of the conversion effort went into converting the many, many string-type handling words to the new dual-behaviour format. None of it was difficult, only tedious.

Recognisers

Recognisers are currently being promoted as a way to provide a user-extensible and user-definable text interpreter. While this is a worthy goal, it isn't enough on its own for systems that already provide a good set of facilities. However, realising that software development tools are not part of the tech industry but of the fashion industry, we bowed to the wind. The recogniser proposals are fluid and not enough people have implemented a heavy application using them.

Instead of treating an interpreter as a tool for finding words, numbers and undefined actions, recognisers provide a list or table or chain of parsers that identify a particular type of element. Once the type has been identified, type data is passed to one of three processing elements for interpretation, compilation or postponing.

The parser may return more than one form of type data. For example, a word recogniser can return separate type data for normal, immediate and dual-behaviour words. In the example below the words starting with **`r:`** are three-element type tables holding the interpretation, compilation and postpone xts.

```
: rec-find \ addr u -- xt r:word | r:fail
\ *G Searches a word in the search order (wordlist stack).
\ ** The xref utility code is contained inside the dictionary
\ ** search code.
  search-context dup 0= if
    drop r:fail
  else
    0< if                   \ -- xt
      dup ndcs?
      if  r:ndcs  else  r:word  then
    else
      r:immediate
    then
  then ;
```

Similarly, you can provide one or more parsers for numeric literals. It's a matter of taste and existing code. If the list of parsers is made extensible, additional word types and literal types can be added at will, and just as importantly, can be removed at will. This facility, together with disciplined use of wordlists and vocabularies is important to enable us to cope with multiple OOP package.

The various recogniser proposals can be found at:

http://amforth.sourceforge.net/pr/Recognizer-rfc-D.pdf
http://amforth.sourceforge.net/pr/Recognizer-rfc-C.pdf
http://amforth.sourceforge.net/pr/Recognizer-rfc-B.pdf
http://amforth.sourceforge.net/pr/Recognizer-rfc.pdf

Experience

In the build of VFX Forth, the only issue was the partitioning of the returned type data. Because we already had an integrated integer handler, we made no distinction between integer types. However, the various floating point packages are installed separately.

We then converted two of the OOP packages supported by MPE. CIAO (C Inspired Active Objects) was the MPE OOP package designed to ease integration with Windows and C++. ClassVFX is the OOP package used by Construction Computer Software in their Candy application:
https://constructioncomputersoftware.com/solutions/solution-candy
The Candy application consists of 1.34 million lines of Forth source code.

The OOP ports revealed that I do not yet understand how to apply **POSTPONE** actions to the result of the dotted notations used by both packages. The most recent set of proposals RFC-D above proposes that the postpone action can be formalised so that the same action can be used by all **POSTPONE** actions. It is just too early to believe that all compound parsers will be able to work this way. One claim for this approach is that it saves memory. We tested this in the VFX kernel and found that the unified action implementation saved 50 bytes in a system of 250k bytes and more. With base-level desktop systems starting with 1Gb of RAM, saving 50 bytes is not a good rationale for standardisation of implementation. We have not yet moved recognisers into our embedded kernel.

The latest recogniser proposal depends on a “stack” proposal that crept in late. That proposal is inadequate for MPE’s OOP requirements as we need to add new parsers at both ends of the tables.

In our recogniser experience to date, recognisers have caused no problems at all and have enabled us to remove a hook or two.

Conclusions

Dual-behaviour words have caused us no problems except for finally having to get rid of any state-smart words in all applications. We have found the notation described in past papers to be easy to use and understand.

Recognisers do enable user-extension of the text interpreter, but the proposals are not yet ready for standardisation. The main reason for this is that although a considerable number of systems have implemented what has been in the proposals, very few systems have pushed the boundaries beyond words and numeric literals. A few string literal proposals have been made, but these little more than handling address and length pairs.

Acknowledgements

Anton Ertl has tested my understanding of Forth standards for many years.

Bernd Paysan has confirmed my belief that the ingenuity of Forth programmers to break common belief must never be underestimated.

My belief that all standards contain bugs has sustained me over many years.

Construction Computer Software encouraged and sponsored the Community edition of VFX v5.