Forth - The New Synthesis

progress report

disaggregating the stacks and memory

EuroForth'21 conference 2021-09-12

Ulrich Hoffmann

uho@

.de

Forth the New Synthesis

Forth

disaggregating

synthesizing

Latest work

- investigate in input and output
  - connection between host and target
    - communicating commands between host and target
    - screens
      - do not need to be 1kB BLOCKs   form-feed separated files
      - b n l list load   can work as usual

Current work

- playing with unicode
- disaggregating stacks
- disaggregating memory

playing with Unicode

- Browsing mathematical Unicode symbols, maybe arrows are nice:

```
    SYNONYM → TO            5 VALUE x       42 → x    x emit
    SYNONYM S→D  S>D        42 S→D D.
    SYNONYM ½· 2/           line-width ½·
    SYNONYM →BODY >BODY     ' eggs →BODY ...
```

- Greek letters:

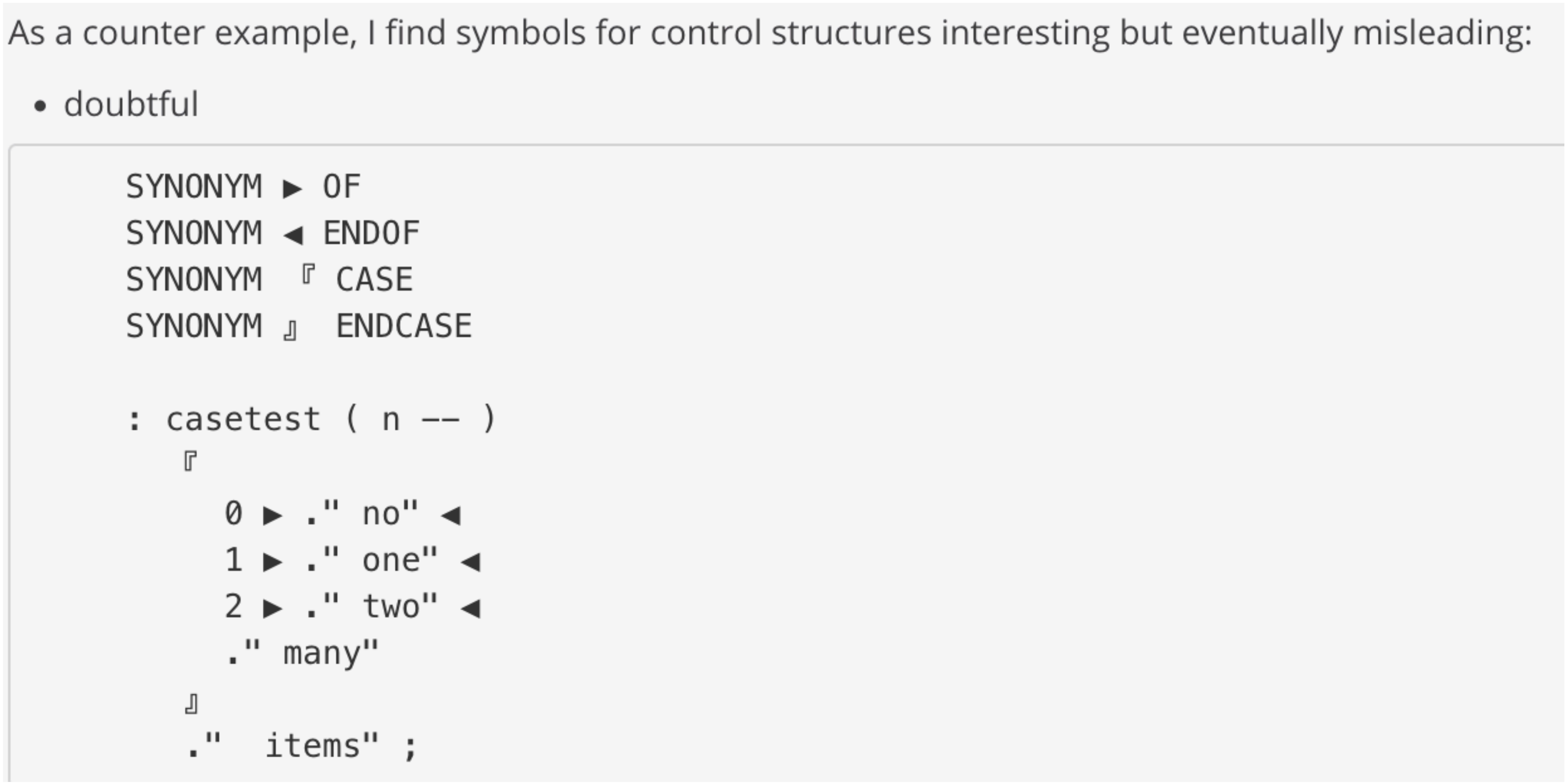
```
    100 CONSTANT Δt          ...   Δt  ms  ...
```

- Or single symbols where we now have symbol sequences:

```
    SYNONYM ≤ <=          ... x 10 ≤ IF ...
    SYNONYM ≠ <>          .... x 45 ≠ IF ...
```

But in general I think you have to be careful using symbols as they best need to have a commony accepted meaning.

playing with Unicode



As a counter example, I find symbols for control structures interesting but eventually misleading:

- doubtful

```
    SYNONYM ▶ OF
    SYNONYM ◀ ENDOF
    SYNONYM 『 CASE
    SYNONYM 』 ENDCASE

    : casetest ( n -- )
       『
          0 ▶ ." no" ◀
          1 ▶ ." one" ◀
          2 ▶ ." two" ◀
          ." many"
       』
       ."  items" ;
```

Disaggregating the Stacks

- data stack and return stack are used for different purposes in different situations.

- disaggregating the stacks means separating these purposes and look at them in isolation.

Disaggregating the Stacks

	Interpreting	Compiling	Executing	comment
Data Stack	parameter passing		parameter passing	
	(unsigned) integers		(unsigned) integers	
	characters		characters	
	floats		floats	
	addresses		addresses	
		control flow		BEGIN IF ...
		compiler security		: ;
		constant folding		
Return Stack	internal return addresses	return addresses	return addresses	
			temporary storage	>R R> R-ALLOT
			loop parameters	DO LOOP
			exception frames	CATCH THROW
			locals	>X X X!

Disaggregating the Stacks

- data stack and return stack are used for different purposes in different situations.

- disaggregating the stacks means separating these purposes and look at them in isolation.

Disaggregating the Stacks

- interferences of the the different purposes lead to restrictions such as:

    - no passing of parameters to definitions at compile time (interference of control flow/compiler security and parameter passing)

    - no use of >R R> across DO-LOOP-boundaries (interference of temporary storage usage and loop parameters)

    - no use of >R R> across definitions (interference of temporary storage and return addressses).

    - specialized stack operators to deal with floating point numbers on the return stack (FDUP, FSWAP, swap cell and float)

Disaggregating the Stacks

Separate stacks for each purpose
Possible disaggregations are

- split data stack into
    - a separate stack for parameter passing that holds (unsigned) integers, characters and also addresses
    - a separate floating point stack for holding floating point numbers (the route Forth-200x went)
    - a separate control flow stack for managing control structures
    - a seperate object stack for handling references to data structures and objects

- split the return stack into
    - a seperate stack for return addreses
    - a seperate stack for temporary data (>R R> R-ALLOT)
    - a seperate stack for loop parameters (DO LOOP)
    - a seperate stack for exception handling (CATCH THROW)
    - a seperate stack for local variables

Disaggregating the Memory

```
: Buffer: ( u -- )
   Create allot ;

: Buffer: ( u -- )
   here swap allot \ RAM  { c0 | ... | cu-1 }
   Create ,         \ ROM  { 'rom }
  Does> ( -- addr ) @
;

: Buffer: ( u -- )
   here swap allot \ RAM
   Constant \ ROM
;
```

<BUILDS

Questions?