preForth

ein minimaler bootstrap-fähiger Forth-Kern

Ulrich Hoffmann <uho@xlerb.de>

Überblick

Bootstrapping Forth

- Einleitung
- preForth
- simpleForth
- interaktives Forth
- Fazit

Forth ist ***words - stacks - blocks***

Jeff Fox

- EuroForth 2016

  Implementing the Forth Inner Interpreter in High Level Forth

- Forth 2017

  Stack der Stacks Strings auf dem Stack

- EuroForth 2017

  handler based outer interpreter

Forth ist ***words - stacks - blocks***

Jeff Fox

- Forth überall (so viel wie möglich)
- bootstrap-fähiges, sich selbst generierendes System
- vollständige Transparenz
- einfach zu verstehen
- auf der Suche nach der Einfachheit
- biologische Analogie
- Kann Forth aus weniger als Forth entstehen?

preForth

- Kann Forth aus weniger als Forth entstehen?
- Was kann man weglassen?
  - kein DOES>
  - kein BASE
  - kein STATE
  - keine formatierte Zahlenausgabe <# # #>
  - kein CATCH/THROW

preForth

- Kann Forth aus weniger als Forth entstehen?
- Was kann man sonst noch weglassen?
  - keine immediate Worte, d.h.
    - keine Kontrollstrukuren IF ELSE THEN BEGIN WHILE REPEAT UNTIL
  - keine definierenden Worte - außer :
  - kein Speicher @ ! CMOVE ALLOT ,
  - kein input stream
  - kein dictionary, kein EXECUTE oder EVALUATE
  - nicht interaktiv

preForth

- Was bleibt dann noch über?
  - Stack
  - Returnstack
  - Nur ?exit und Rekursion als Kontrollstrukturen
  - :-Definitionen
  - optional Tail Call Optimierung
  - Ein- und Ausgabe via KEY/EMIT
  - dezimale positive und negative Zahlen (eine Zelle)
  - Zeichen-Literale in 'x'-Notation
  - dezimale Zahlenausgabe (eine Zelle)

preForth-Programme

Wie sehen sie aus?

```
: countdown ( n -- )
   cr dup .
   ?dup 0= ?exit
   1-  tail countdown ;
```

```
5 countdown
```

```
5 4 3 2 1 0
```

preForth-Programme

Wie sehen sie aus?

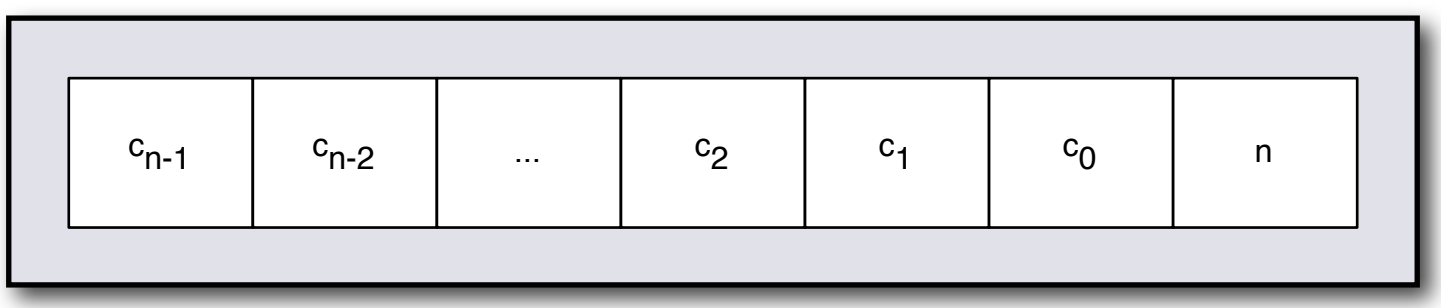
```
: dashes ( n -- )
  ?dup 0= ?exit
    '-' emit 1- tail dashes ;
```

```
5 dashes

-----
```

preForth-Programme

Wie sehen sie aus?

```
\ show displays topmost string

: show ( S -- )
   ?dup 0= ?exit  swap >r 1- show
   r> emit ;
```



preForth-Programme

Wie sehen sie aus?

```
: ."Hello,_world!" ( -- )
    'H' 'e' 'l' 'l' 'o' ',' bl
    'w' 'o' 'r' 'l' 'd' '!' 13 show ;
```

Hello world!

preForth-Operationen für Stack-Strings

- **`_dup ( S -- S S )`**
- **`_swap ( S1 S2 -- S2 S1 )`**
- **`_drop ( S -- )`**
- **`_show ( S -- )`**
- Muster
  - **`dup pick  ( S -- c )`** erstes Zeichen
  - **`swap 1+ ( S1 c -- S2 )`** Zeichen anfügen

Pick und Roll ?!

```
: pick ( xn-1 ... x0 i -- xn-1 ... x0 xi )
    over swap ?dup 0= ?exit nip swap
    >r 1- pick r> swap ;

: roll ( xn-1 ... x0 i -- xn-1 ... xi-1 xi+1 ... x0 xi )
    ?dup 0= ?exit swap >r 1- roll r> swap ;
```

```
: ?dup ( x -- x x | 0 )
    dup dup ?exit drop ;
```

Primitives

- Forth überall (so viel wie möglich)
- eine Basis muss es geben:
  - 13 Primitives:

```
emit key
dup swap drop
0< -
?exit
>r r>
nest unnest
lit
```

Defintion von Primitives

Formulierung in der Plattform Zielsprache (hier i386-Asm)

```
code ?exit ( f -- )
        pop eax
        or eax, eax
        jz qexit1
        mov esi, [ebp]
        lea ebp,[ebp+4]
qexit1: next
;
```

Beschreibung von Zielcode

Formulierung in der Plattform Zielsprache (hier i386-Asm)

```
prefix
format ELF

...

macro next   {
        lodsd
        jmp dword [eax]
}
...
;
```

pre

prelude

prefix

preamble

preformatted

preForth compiler

- Akzeptiert preForth-Programm von stdin
- Schreibt Plattform-Programme nach stdout
  - hier i386-Assembler
  - weitere Backends sehr einfach (C, geplant x64, stm8, NIGE)
- Selbst in preForth formuliert
- Kann sich selbst reproduzieren
- Erster Bootstrap via gForth oder SwiftForth
- Maschinencode wird über Plattform-Assembler erzeugt

preForth compiler

- Outer interpreter und compiler basieren auf Handlern
- Handler **`( S -- i*x 0 | S )`**

```
\ ?'x' detects and compiles a character literal
: ?'x' ( S -- 0 | S )
       dup 0= ?exit
       dup 3 - ?exit
       over   ''' - ?exit
       3 pick ''' - ?exit
       2 pick >r _drop r>
       ,lit 0 ;
```

- Handler werden in :-Definitionen kombiniert.

preForth compiler

- Handler werden in :-Definitionen kombiniert.
- Compiler loop:

```
: ] ( -- )
    token                 \ get next token
    \ run compilers
    ?; ?dup 0= ?exit      \ ;  leave compiler loop
    ?\                    \ comment
    ?tail                 \ marked as tail call
    ?'x'                  \ character literal
    ?lit                  \ number
    ?word                 \ word
    _drop  tail ] ;       \ ignore unhandled token and cycle
```

generierter Plattform-Code

?exit

```
; ?exit
_Qexit:  DD _QexitX
_QexitX:pop eax
        or eax, eax
        jz qexit1
        mov esi, [ebp]
        lea ebp,[ebp+4]
qexit1: next
```

?dup

```
; ?dup
_Qdup:   DD _nest
_QdupX:
         DD _dup
         DD _dup
         DD _Qexit
         DD _drop
         DD _unnest
```

Bootstrapping preForth

demo

simpleForth

- preForth ist turing-vollständig.

  Ein volles Forth in preForth zu formulieren ist möglich...
  ... aber es ist relativ mühsam.

- preForth erweitern: simpleForth

simpleForth

- simpleForth ist wie preForth
- preForth $\subset$ simpleForth

- zusätzlich:
  - Kontrollstrukturen: IF ELSE THEN BEGIN WHILE REPEAT UNTIL
  - Definitionen mit und ohne Header im generierten Code
  - Speicher: @ ! c@ c! allot c, ,
  - variable constant
  - ['] execute
  - immediate Definitionen

Bootstrapping Forth

- volles Forth ("*Forth*") in simpleForth
- Work-in-Progress
  - mit neuen Forth-Techniken experimentieren
    - Handler basierter Text-Interpretierer
    - NDCS
    - Speicherverwaltung
    - ...
- Derzeit klassisches ITC-System
  als proof of concept

Forth

- Beispiel:

```
: sqr ( x -- x² )
    dup * ;

: sqrt ( x² -- x )
    1 BEGIN  2dup /  over - 2 /
        dup
      WHILE
        +
      REPEAT drop nip ;

: pyt ( a b -- c )
    sqr swap sqr +  sqrt ;
```

Bootstrapping Forth

demo

aktuelle Beobachtung

- "doppelte" Beschreibung
  1. für entstehendes Forth-Image
  2. für interaktives System
     - Kontrollstrukturen
     - Header

Fazit

- Forth überall (so viel wie möglich)

computational clay

- bootstrap-fähiges, sich selbst generierendes System
- vollständige Transparenz
- einfach zu verstehen

- Kann Forth aus weniger als Forth entstehen?

Ja - mit preForth ☺

✔ ✔

Forth ist ***words - stacks - blocks***

Jeff Fox