Forth, The New Synthesis:

Growing Forth with preForth and `seedForth`

Ulrich Hoffmann

uho@ .de

https://github.com/uho/preForth

Overview

- Introduction: Forth, the New Synthesis
  - family of minimalistic stack based languages
- the ICE concept
- `seedForth`
  accepting tokenized source code
- summary and future work
- Q&A

Forth, the new synthesis

The new synthesis is an ongoing effort

- to understand
  - the general foundation of computation
  - especially the basic principles of Forth
- to form the basis of a new modern Forth

Forth, the new synthesis

Our guidelines are

- Forth everywhere (as much as possible)
- bootstrap-capable self-generating system
- completely transparent
- simple to understand
- quest for simplicity
- biological analogy
- disaggregation and recombination

We build a family of minimalistic stack based languages in order to study their essence.

family of minimalistic stack based languages

	preForth	**seedForth**
purpose	bootstrap seedForth	application plattform
accepted source code	text based	token based
stacks	parameter/return	parameter/return
LOC	<500	<550
# of primitives	13	31
recursive functions	✔	✔
random access memory	none	✔
string handling	on stacks	in memory
function definitions	platform and Forth	Forth
control structures	(tail) recursion, conditional exit	(tail) recursion, conditionals, loops
easily retargetable	✔	✔
input/output	character/int i/o stdin/stdout	character i/o stdin/stdout
data types	character/int	character/int/address
interpreter	none	✔
compiler	✔	✔

ICE concept

Moore 1999

intermix

- **I**nterpret
- **C**ompile
- **E**xecute

- Language property of Forth, Lisp, Python
  - define a function, it gets **compiled**
  - invoke a function, its arguments get **interpreted**
  - and the function will be **executed**

  - the function's side effect or its result can be used in the remaining program

  - executing functions during compilation can generate code

ICE concept

```
: erase ( c-addr u -- )                  \ compile
   bounds ?DO 0 I c! LOOP ;

1024 Constant bufsize                    \ interpret
Create buf  bufsize allot

buf bufsize erase                        \ execute
```

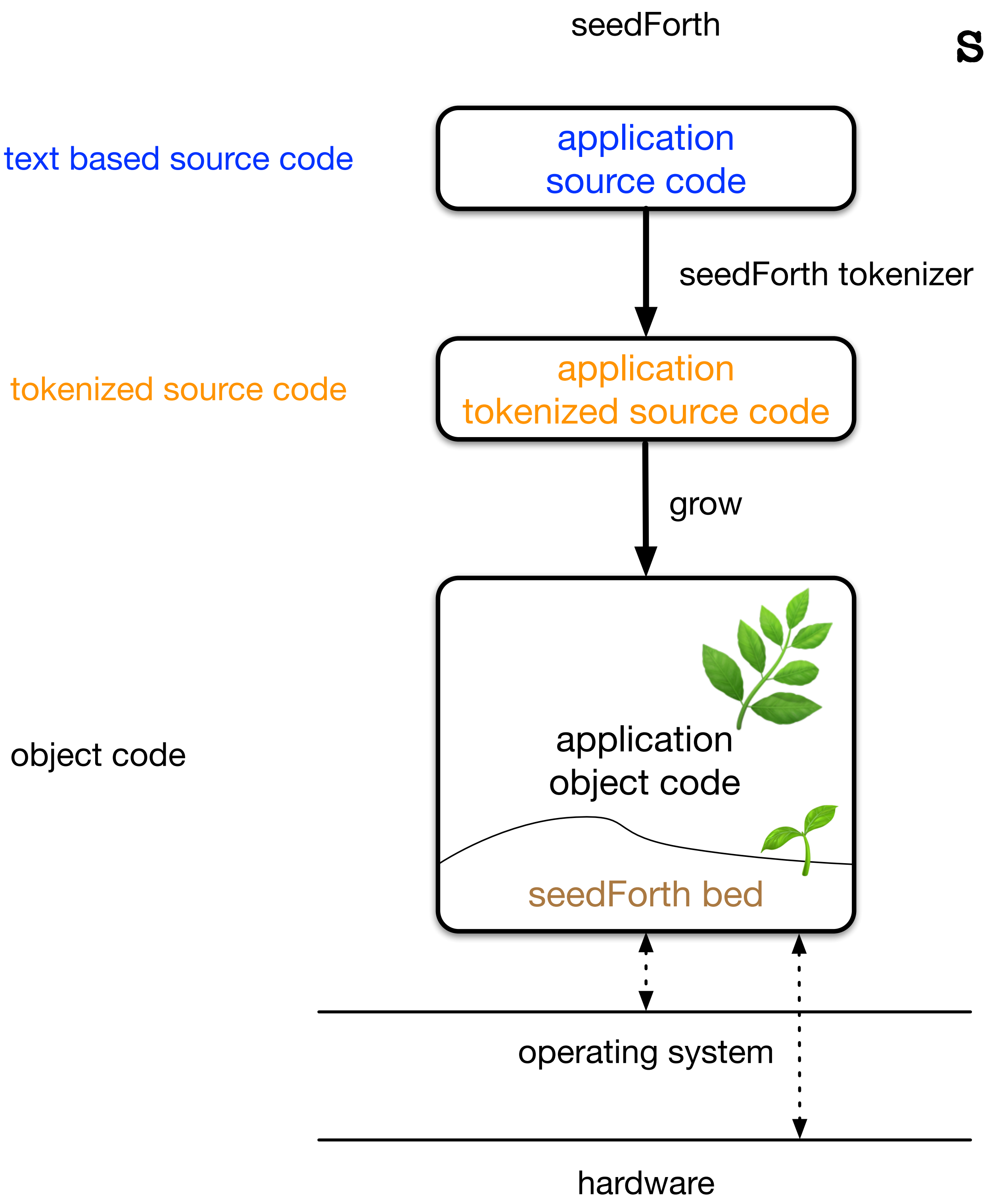
seedForth

`seedForth`

- *accepts source code in tokenized form*
- the `seedForth bed` is just 550 LOC
- is extensible by function (aka *colon*) definitions
- follows the ICE principle and so provides
    - a **compiler** that compiles definitions
    - an **interpreter** that can **execute** definitions
- is extended by application code to create apps
- can be extended to a full-featured interactive Forth

- current implementations for i386 and AMD64

seedForth bed

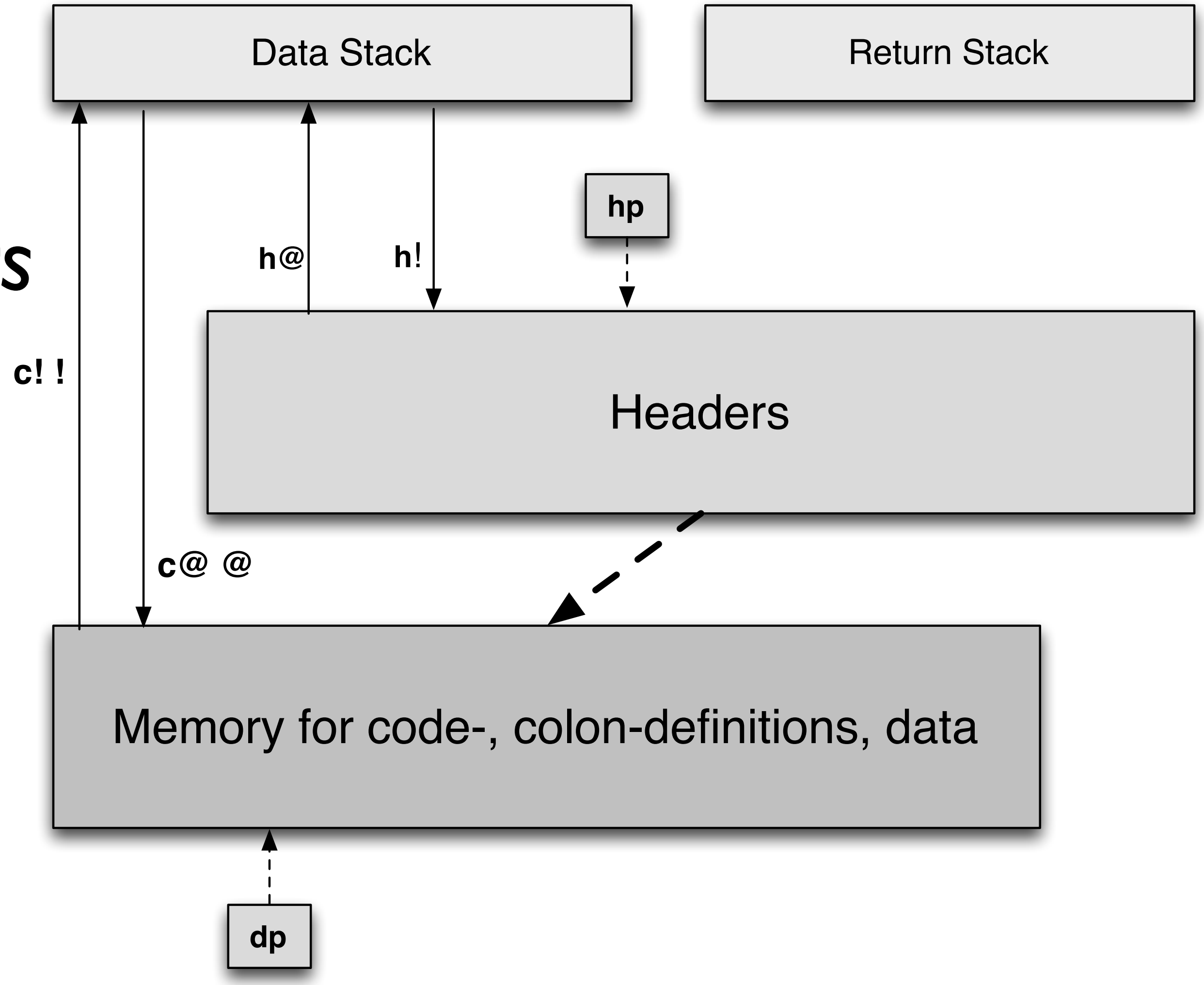


- very easy to adapt to new hardware (e.g. IoT devices)
- bring up time:
  half a day
- all above seed bed can be left untouched
- minimal memory footprint (i386: 2KB)
- easy to understand completely from top to bottom

seedForth architecture

simplify names:

names are just numbers



seedForth virtual machine

- data (parameter) stack, return stack
- addressable memory for code, function definitions, data
- headers: array mapping word indices to start addresses

seedForth bed words

```
(  0 $00 ) Token bye       Token prefix1     Token prefix2    Token emit
(  4 $04 ) Token key       Token dup         Token swap       Token drop
(  8 $08 ) Token 0<        Token ?exit       Token >r         Token r>
( 12 $0C ) Token -         Token exit        Token lit        Token @
( 16 $10 ) Token c@        Token !           Token c!         Token execute
( 20 $14 ) Token branch    Token ?branch     Token negate     Token +
( 24 $18 ) Token 0=        Token ?dup        Token cells      Token +!
( 28 $1C ) Token h@        Token h,          Token here       Token allot
( 32 $20 ) Token ,         Token c,          Token fun        Token interpreter
( 36 $24 ) Token compiler  Token create      Token does>      Token cold
( 40 $28 ) Token depth     Token compile,    Token new        Token couple
( 44 $2C ) Token and       Token or          Token sp@        Token sp!
( 48 $30 ) Token rp@       Token rp!         Token $lit       Token num
( 52 $34 ) Token um*       Token um/mod      Token unused     Token key?
( 56 $38 ) Token token     Token usleep      Token hp
```

```
: interpreter ( -- )
   token execute   tail interpreter ;
```

```
: compiler ( -- )
   token ?dup 0= ?exit ?lit
   compile,   tail compiler ;
```

seedForth tokenizer

- function names map to *single* tokens (function numbers)
- number and character literals map to token *sequences*
- control structures map to token *sequences*
- **:** starts a new function definition and invokes compiler
- **;** stops compiler and ends function definition

hello.seedsource

```
PROGRAM hello.seed
'H' emit  'e' emit  'l' dup emit emit  'o' emit  10 emit

 : 1+ ( x1 -- x2 ) 1 + ;

'A' 1+  emit   \ outputs B
END
```

hello.seed

```
00000000  33 04 48 0d 03 33 04 65  0d 03 33 04 6c 0d 05 03  |3.H..3.e..3.l...|
00000010  03 33 04 6f 0d 03 33 04  0a 0d 03 22 33 04 01 0d  |.3.o..3...."3...|
00000020  17 0d 00 33 04 41 0d 3b  03 00                    |...3.A.;..|
```

seedForth tokenizer

- function names map to *single* tokens (function numbers)
- number and character literals map to token *sequences*
- control structures map to token sequences
- **:** starts a new function definition and invokes compiler
- **;** stops compiler and ends function definition

hello.seedsource

```
PROGRAM hello.seed
'H' emit  'e' emit  'l' dup emit emit  'o' emit  10 emit

 : 1+ ( x1 -- x2 ) 1 + ;

'A' 1+  emit   \ outputs B
END
```

```
Hello
B
```

hello.seed

```
00000000  33 04 48 0d 03 33 04 65  0d 03 33 0           3  |3.H..3.e..3.l...|
00000010  03 33 04 6f 0d 03 33 04  0a 0d 03 22 33 04 01 0d  |.3.o..3....."3...|
00000020  17 0d 00 33 04 41 0d 3b  03 00                    |...3.A.;..|
```

seedForth tokenizer

- *control structures map to token sequences*
- **BEGIN** ... condition **UNTIL**   simple loop
- **here** puts the memory address where code is generated on parameter stack
- **,** lays down the value on the parameter stack at **here**

```
BEGIN ( -- addr ) maps to the token sequence  bye  here  compiler
                                              $00  $1E   $24

UNTIL ( addr -- ) maps to the token sequence  ?branch  bye ,   compiler
                                              $15      $00 $20 $24
```

```
PROGRAM countdown.seed
: .digit ( u -- ) '0' + emit ;
: countdown ( u -- ) BEGIN 1 - dup .digit dup 0= UNTIL  drop ;
10 countdown
END
```

```
00000000  22 33 04 30 0d 17 03 0d  00 22 00 1e 24 33 04 01  |"3.0....."..$3..|
00000010  0d 0c 05 3b 05 18 15 00  20 24 07 0d 00 33 04 0a  |...;.... $...3..|
00000020  0d 3c 00                                          |.<.|
```

seedForth tokenizer

- *control structures map to token sequences*
- **BEGIN** ... condition **UNTIL** simple loop
- **here** puts the memory address where code is generated on parameter stack
- **,** lays down the value on the parameter stack at **here**

```
BEGIN ( -- addr ) maps to the token sequence  bye   here   compiler
                                              $00   $1E    $24

UNTIL ( addr -- ) maps to the token sequence  ?branch  bye ,    compiler
                                              $15      $00 $20  $24
```

```
PROGRAM countdown.seed
: .digit ( u -- ) '0' + emit ;
: countdown ( u -- ) BEGIN 1 - dup .digit dup 0= UNTIL  drop ;
10 countdown
END
```

```
9876543210
```

```
00000000  22 33 04 30 0d 17 03 0d  00 22 00 1                 ....."..$3..|
00000010  0d 0c 05 3b 05 18 15 00  20 24 07 0d 00 33 04 0a  |...;.... $...3..|
00000020  0d 3c 00                                          |.<.|
```

seedForth grows

extensions for application development

- ✓ dynamic memory allocation with **allocate**, **resize** and **free**
- ✓ defining words including **DOES>** (**Definer**)
- ✓ compiling words (control structures, **Macro**)
- ✓ exception handling (**catch**, **throw**)
- ✓ cooperative multitasking (**pause**, **activate**)
- ✓ quotations (**[:** and **;]**)
- the tokenizer expressed in seedForth
- ...

extensions towards a full-featured interactive Forth

- ✓ headers with dictionary search
- ✓ text interpreter and compiler that work on text source
- ✓ optimizers: inline, peephole, constant folding
- a Forth assembler for the target platform and additional primitives
- OOP
- file and operating system interface
- access to hardware
- ...

seedForth/interactive

summary and future work

The New Synthesis

The ICE concept: Interpret, Compile, Execute

`seedForth`

- accepts tokenized source code
- names are just number indices into the header array
- grow the seedForth bed to build applications
- extensible to a complete, interactive Forth
- easy to understand from top to bottom

future work

- extend seedForth/interactive to support ANS-Forth
- IoT targets
- "New Synthesis" the book

Q&A