GNU Mes Maxwell Equations of Software

janneke@gnu.org

FOSDEM'17

2017-02-05

Outline

Mes: Full Source Bootstrapping

- Where do compilers come from?
- Who compiled the compiler?
- Chicken and Egg

WTF

Mes: Full Source Bootstrapping

- Where do compilers come from?
- Who compiled the compiler?
- Chicken and Egg

WTF

Mes: Full Source Bootstrapping

- Where do compilers come from?
- Who compiled the compiler?
- Chicken and Egg

WTF

Mes is a strategy

- NOT a goal in itself – only a means or proof of concept
- NOT a general purpose Scheme – close to R6RS
- NOT an alternative for Guile – reuse Guile modules

Inspiration: what do you want?

Inspiration: what do you want?

Meaning, Autonomy, Co-Creation, Self-Realization

- Discovering, Hacking, Motivating, Playing

Inspiration: what do you want?

Meaning, Autonomy, Co-Creation, Self-Realization

- Discovering, Hacking, Motivating, Playing

A planet of enlightened beings

- Look inward
- Be happy
- Be helpful

Inspiration: what do you want?

Meaning, Autonomy, Co-Creation, Self-Realization

- Discovering, Hacking, Motivating, Playing

A planet of enlightened beings

- Look inward
- Be happy
- Be helpful

A world where all software is free

- Support GNU
- Create free software

Inspiration: when do you want it?

Inspiration: when do you want it?

NOW!!!

Inspiration

To finally run GNU

- GuixSD: GNU in the flesh

1984 Four Software Freedoms: GNU GPL

The freedom to

- 0 run the program as you wish, for any purpose
- 1 study how the program works, and change it if you wish
- 2 redistribute copies so you can help your neighbor
- 3 share copies of your modified versions with others

– Richard M. Stallman

2013 Debian's reproducible-builds.org

Verifiable path: source -> binary

Reproducible builds are a set of software development practices that create a verifiable path from human readable source code to the binary code used by computers.

Does this binary come from the given source?

- Always different binary. . . dunno?
- Same binary
    - Always good, always bad?

2015 GuixSD Reproducible builds: a means to an end

A technical means to an end

- guarantee user autonomy and safety
    - GNU+GuixSD: fully free distro
    - NixOS: fully isolated build environment
    - NixOS: full list of dependencies
    - reproducible builds: bit-for-bit identical binaries

– Ludovic Courtès

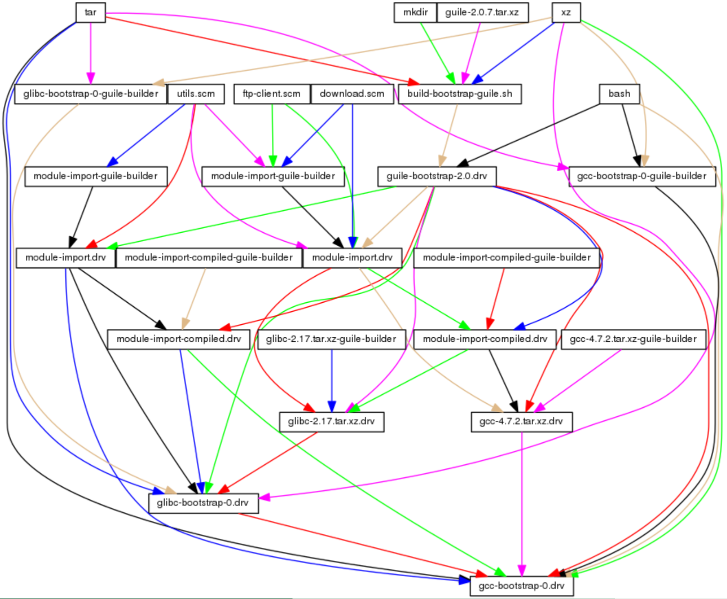
2016 From GuixSD to Mes: The bootstrap binaries

GuixSD ... source

- source/binary transparency
- all is built from source
- EVERYTHING
- starting from the ... bootstrap binaries

The distribution is fully "bootstrapped" and "self-contained": each package is built based solely on other packages in the distribution.
The root of this dependency graph is a small set of "bootstrap binaries", provided by the '(gnu packages bootstrap)' module. For more information on bootstrapping, *note Bootstrapping::.
[2010]: Eelco Dolstra, Andres Löh, and Nicolas Pierron described sources of non-determinism in their 2010 JFP paper about NixOS

GuixSD bootstrap graph


tar
mkdir
guile-2.0.7.tar.xz
xz
glibc-bootstrap-0-guile-builder
utils.scm
ftp-client.scm
download.scm
build-bootstrap-guile.sh
bash
module-import-guile-builder
module-import-guile-builder
guile-bootstrap-2.0.drv
gcc-bootstrap-0-guile-builder
module-import.drv
module-import-compiled-guile-builder
module-import.drv
module-import-compiled-guile-builder
module-import-compiled.drv
glibc-2.17.tar.xz-guile-builder
module-import-compiled.drv
gcc-4.7.2.tar.xz-guile-builder
glibc-2.17.tar.xz.drv
gcc-4.7.2.tar.xz.drv
glibc-bootstrap-0.drv
gcc-bootstrap-0.drv

GuixSD bootstrap tarballs

```
$ du -schx $(readlink $(guix build bootstrap-tarballs)/*)
2.1M /gnu/store/mzk1bc3pfrrf4qnfs3zkj5ch83srnvpx-binutils-stati
16M /gnu/store/jddviycivycfhaqahqff6n18y9w46gpz-gcc-stripped-ta
1.7M /gnu/store/x5zrmh820yc054w00cy00iixwghmly2y-glibc-strippec
3.1M /gnu/store/znsf5d7xbqkp4rrjgzsklmwmms8m5i3m-guile-static-s
5.7M /gnu/store/myfikfgx74dzlm3lc217kchxnckri5qq-static-binarie
28M total
$ for i in $(readlink $(guix build bootstrap-tarballs)/*);\
  do sudo tar xf $i; done
$ du -schx *
125M bin
13M include
18M lib
43M libexec
4.3M share
202M total
```

Inspiration

To finally run GNU

- GuixSD: GNU in the flesh

Bootstrap binaries: source all the way down?

- OriansJ: self-hosting hex assembler

Inspiration

To finally run GNU

- GuixSD: GNU in the flesh

Bootstrap binaries: source all the way down?

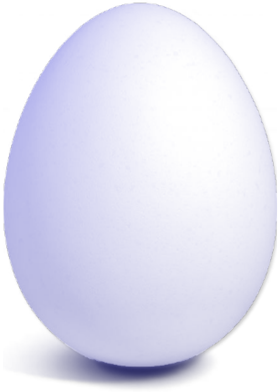
- OriansJ: self-hosting hex assembler

The computer revolution hasn't happened yet

- Alan Kay

The computer revolution is very new, and all of the good ideas have not been universally implemented

Bootstrapping: Chicken and Egg



Inspiration

To finally run GNU

- GuixSD: GNU in the flesh

Bootstrap binaries: source all the way down?

- OriansJ: self-hosting hex assembler

The computer revolution hasn't happened yet

- Alan Kay

The computer revolution is very new, and all of the good ideas have not been universally implemented

LISP as the Maxwell's Equations of Software

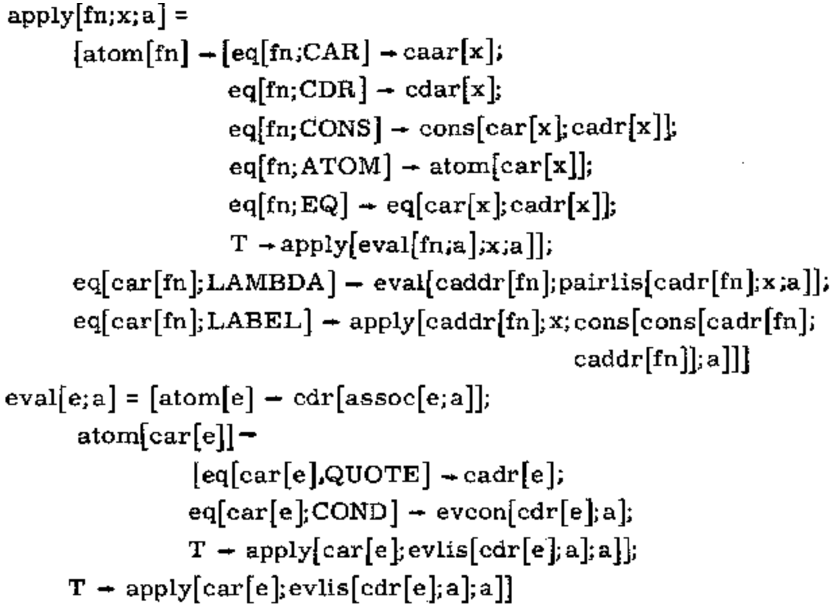
That was the big revelation to me when I [..] finally understood that the half page of code on the bottom of page 13 of the Lisp 1.5 manual was Lisp in itself. These were "Maxwell's Equations of Software!"

Chicken and Egg

- the shortest path from hex to gcc
- using Maxwell's Equations of Software

Bootstrapping: Chicken and Egg

LISP-1.5 John McCarthy: page 13



```
apply[fn;x;a] =
      [atom[fn] → [eq[fn;CAR] → caar[x];
                   eq[fn;CDR] → cdar[x];
                   eq[fn;CONS] → cons[car[x];cadr[x]];
                   eq[fn;ATOM] → atom[car[x]];
                   eq[fn;EQ] → eq[car[x];cadr[x]];
                   T → apply[eval[fn;a];x;a]];
      eq[car[fn];LAMBDA] → eval[caddr[fn];pairlis[cadr[fn];x;a]];
      eq[car[fn];LABEL] → apply[caddr[fn];x;cons[cons[cadr[fn];
                                                        caddr[fn]];a]]]
eval[e;a] = [atom[e] → cdr[assoc[e;a]];
       atom[car[e]] →
                [eq[car[e];QUOTE] → cadr[e];
                eq[car[e];COND] → evcon[cdr[e];a];
                T → apply[car[e];evlis[cdr[e];a];a]];
      T → apply[car[e];evlis[cdr[e];a];a]]
```

Eval/Apply

- core
  - apply
  - eval
- helpers
  - assoc
  - pairlis
  - evcon
  - evlis
- primitives
  - atom
  - car
  - cdr
  - cons
  - eq

LISP-1.5 in Guile Scheme: APPLY

```
(define (apply fn x a)
  (cond
   ((atom fn)
    (cond
     ((eq fn CAR)  (caar x))
     ((eq fn CDR)  (cdar x))
     ((eq fn CONS) (cons (car x) (cadr x)))
     ((eq fn ATOM) (atom (car x)))
     ((eq fn EQ)   (eq (car x) (cadr x)))
     (#t           (apply (eval fn a) x a))))
   ((eq (car fn) LAMBDA)
                   (eval (caddr fn) (pairlis (cadr fn) x a)))
   ((eq (car fn) LABEL)
                   (apply (caddr fn) x (cons (cons (cadr fn)
                                                   (caddr fn))
                                             a)))))
```

LISP-1.5 in Guile Scheme: EVAL

```
(define (eval e a)
  (cond
   ((atom e) (cdr (assoc e a)))
   ((atom (car e))
    (cond ((eq (car e) QUOTE) (cadr e))
          ((eq (car e) COND)  (evcon (cdr e) a))
          (#t                 (apply (car e)
                                     (evlis (cdr e) a) a))))
   (#t       (apply (car e) (evlis (cdr e) a) a))))
```

LISP-1.5 in Scheme: ASSOC, PAIRLIS, EVCON, EVLIS

```
(define (assoc x a)
  (cond ((eq (caar a) x) (car a))
        (#t (assoc x (cdr a)))))

(define (pairlis x y a)
  (cond ((null x) a)
        (#t (cons (cons (car x) (car y))
                  (pairlis (cdr x) (cdr y) a)))))

(define (evcon c a)
  (cond ((eval (caar c) a) (eval (cadar c) a))
        (#t (evcon (cdr c) a))))

(define (evlis m a)
  (cond ((null m) NIL)
        (#t (cons (eval (car m) a) (evlis (cdr m) a)))))
```

LISP-1.5 in C

- closures
- symbols
- specials? `() #t #f *unspecified* *undefined*`
- macros
- syntax-rules
- records
- modules/importing

Garbage/Jam Collector

Abelson & Sussman

With a real computer we will eventually run out of free space in which to construct new pairs.(1)

footnote(1)

This may not be true eventually, because memories may get large enough so that it would be impossible to run out of free memory in the lifetime of the computer. For example, there are about $\{3 \cdot 10^{13}\}$ microseconds in a year, so if we were to 'cons' once per microsecond we would need about 10^{15} cells of memory to build a machine that could operate for 30 years without running out of memory.

C parser: roll your own LALR

Lalr

- minimal ANSI-C parser
  `int main (){puts ("Hello, world!");return 0;}`

C parser: Nyacc

Pros

- full C99 parser
- . . . including C pre-processor
- perspective of building complete C compiler in Guile
- tsunami of enthusiasm and contributors!

C parser: Nyacc

Cons: more TODO for Mes

- keywords
- `define*`, `lambda*`
  - optargs
- exeptions, `catch`, `throw`
  - `call/cc`
- fluids, `with-fluid`
- `syntax-case`
  - André van Tonder's 2006-2007 streak in 14 "commits"
  - psyntax: another bootstrap loop?!
- R7RS's Ellipsis
  - Guile-1.8
- #;-comments
- #||#-comments

C parser: Nyacc

Cons: more TODO for Mes

- Cond supports =>
- Bugfixes
  - Cond now evaluates its test clauses only once
  - Append can also handle one argument
  - For-each now supports 2 list arguments
  - Map now supports 3 list arguments
  - Backslash in string is supported
  - Closure is not a pair
  - All standard characters are supported
- 36 new functions

```
1+, 1-, abs, and=>, append-reverse, ash, char<\=?, char<?,
char>=?, char>?, even?, filter, delete, delq, vector-copy,
fold, fold-right, getenv, iota, keyword->symbol list-head,
list-tail, negative?, odd?, positive?, remove!, remove,
string->number, string-copy, string-prefix?, string=,
```

Timeline

June 19: on bootstrapping: introducing Mes

- LISP-1.5 in Scheme and in C

September 25: on bootstrapping: 2nd status report on Mes

- Scheme primitives in C, closures, macros, 97 tests, LALR
- Produce ELF binary from

```
int main ()
{
  int i;
  puts ("Hi Mes!\n");
  for (i = 0; i < 4; ++i)
    puts ("  Hello, world!\n");
  return 1;
}
```

Timeline

October 23: 0.1 [not announced]

- `let-syntax`, `match`
- compile main.c in 2s (was 1'20")
- add REPL

November 21: 0.2 [not announced]

- psyntax integration, `syntax-case`, `load`

December 12: on bootstrapping: first Mes 0.3 released

- Garbage Collector/Jam Scraper

December 25: Mes 0.4 released

- run Nyacc, PEG, reduced core

Status

core C prototype: 1150 lines

non-essential C sources:

```
210 lib.c
157 math.c
126 posix.c
134 reader.c
627 total
```

Status

tiny-mes.c: 270 lines

- compiles with mescc
- i386-lib: `i386:exit`, `i3886:open`, `i386:read`, `i386:write`
- tiny-libc: `getchar`, `putchar`, `puts`, `strcmp`, `strlen`
- runs

```
Hello tiny-mes!
reading: module/mes/hack-32.mo
MES *GOT MES*
(#\A(#\B))
```

Status

mini-mes.c: 800 lines

- 12kB binary
- 2500 lines assembly
- runs with gcc

```
Hello mini-mes!
reading: module/mes/hack-32.mo
MES *GOT MES*
cells read: 19
symbols: 1
program[10]: (cons(0(1)))
(0 . 1)
```

- compiles with mescc
  - 83 statements skipped

Status

current Guix package

```
01:16:51 janneke@dundal:~/src/mes
$ guix package -f guix.scm
The following package will be upgraded:
   mes 0.4.f84e97fc -> 0.4.f84e97fc /gnu/store/2fsy1cd24pnwkv7a
```

What's next?

- psyntax
    - source or binary?
    - alternative syntax-case?
    - rewrite Nyacc without syntax-case, R7RS-ellipsis?
- call/cc vs eval/apply/evlis?
- merge with Guile?
- compile Guile or compile Gcc?
- prototype? in C
    - move from C to Hex?
    - move from C to [Pre]Scheme

Thanks

Thanks

- John McCarthy
- Richard Stallman
- Eelco Dolstra
- Ludovic Courtès
- Rutger van Beusekom
- Christopher A. Webber

Thanks everyone else

- LISP-1.5
- GNU
- NixOS
- Debian reproducible builds
- GuixSD
- FOSDEM

Connect

- irc freenode.net #guix #guile
- mail guile-user@gnu.org
- git git@gitlab.com:janneke/mes.git