```
    _____
---'   __\_______
           ______)  GNU(*) poke
           __)
          __)
---._______)
```

The extensible editor
for structured binary data

Jose E. Marchesi

Kernel Recipes 2019

(*) approval pending

Disclaimer

This is **fun in progress**

Contents

Motivation

```
# Figure out the file offset of the text
# section in the object file.
text_off=0x$(objdump -j .text -h $objfile \
             | grep \.text | $TR -s ’ ’ \
             | $CUT -d’ ’ -f 7)

...

func_off=$(printf %s $fun | $CUT -d: -f1)
base=$($EXPR $func_off + 0)
probe_off=$((text_off + base + offset))
...
byte=$(dd if=$objfile count=1 ibs=1 bs=1 \
          skip=$probe_off 2> /dev/null)
```

Motivation

- Need to edit object files, among others.
- Scripts break easily, and are a PITA to maintain.
- Format-specific tools are... too specific.
- Decided to hack a general-purpose binary editor in 2017.
- ... **poke** happened after 2 years of work.

Developing the idea

- Took a while.
- From C structs “plus something” to a full-fledged programming language.
- Nice but unsatisfactory existing work: **Datascript** by Godmar Back.
- Unacceptable and simplistic existing work: 010 Editor.
- After many design failures and blind alleys... finally got it right... or so I hope! :D

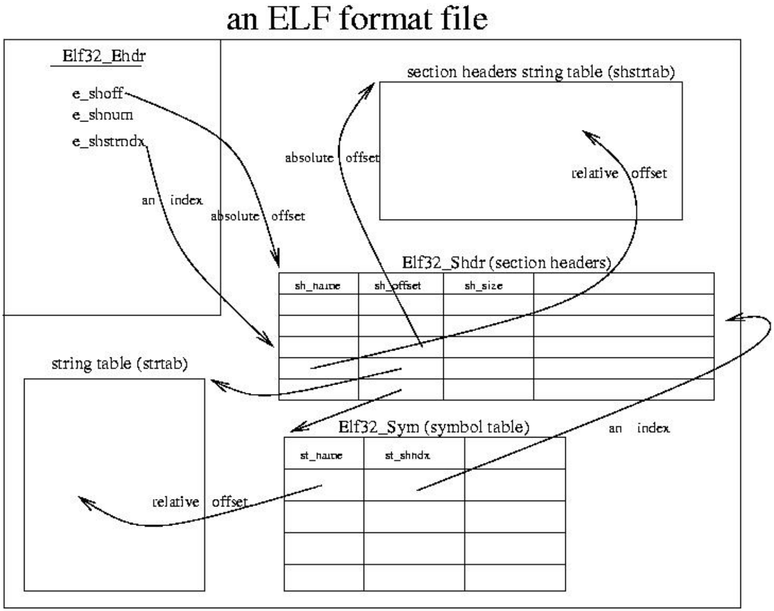
Overview

```
     _____
 ---'   __\_______
            ______)  GNU poke 0.1-beta
            __)
           __)
 ---._______)

Copyright (C) 2019 Jose E. Marchesi.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Powered by Jitter 0.9.0.556-d1e5.
Perpetrated by Jose E. Marchesi.

For help, type ".help".
Type ".exit" to leave the program.
(poke) dump
76543210  0011 2233 4455 6677 8899 aabb ccdd eeff
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000
00000010: 0100 3e00 0100 0000 0000 0000 0000 0000
00000020: 0000 0000 0000 0000 0802 0000 0000 0000
00000030: 0000 0000 4000 0000 0000 4000 0b00 0a00
00000040: 5548 89e5 b800 0000 005d c300 4743 433a
00000050: 2028 4465 6269 616e 2036 2e33 2e30 2d31
00000060: 382b 6465 6239 7531 2920 362e 332e 3020
00000070: 3230 3137 3035 3136 0000 0000 0000 0000
(poke)
```

Demo!

Poking a relocation in an ELF file

Demo!


an ELF format file
Elf32_Ehdr
e_shoff
e_shnum
e_shstrndx
an index
absolute offset
absolute offset
section headers string table (shstrtab)
relative offset
Elf32_Shdr (section headers)
sh_name
sh_offset
sh_size
string table (strtab)
Elf32_Sym (symbol table)
st_name
st_shndx
an index
relative offset

The language - Values

- Integers:

```
10, 0xff, 8UB, 0b1100, 0o777
```

- Strings:

```
"foo\nbar"
""
```

- Arrays:

```
[1,2,3]
[[1,2],[3,4]]
[[1,2,3],[4]]
```

- Structs:

```
struct { name = "Donald␣Knuth", age = 100 }
struct {}
```

The language - Offset values

- The offset problem.
- bytes? bits? both?
- Solution: **united values**.

The language - Offset values

- Named units:

```
8#b
23#B
2#Kb
```

- Numeric units:

```
8#8
2#3
```

- Even better:

```
deftype Packet = struct { int i; long j; }
23#Packet
```

- Operations:

```
OFF +- OFF -> OFF
OFF *  INT -> OFF
OFF /  OFF -> INT
OFF %  OFF -> OFF
```

The language - Offset values

Offsets avoid explicit unit conversions

```
deftype Elf64_Shdr =
  struct
  {
    ...
    offset<Elf64_Xword,B> sh_size;
    ...
  };

...
shdr.sh_size = 10#Elf64_Rela;
```

The language - Simple Types

- Integral types:

  ```
  int<N>
  uint<N>
  ```

- Offset types:

  ```
  offset<INT_TYPE,UNIT>
  ```

- String type:

  ```
  string
  ```

The language - Array Types

- Unbounded:

```
int[]
int[][]
```

- Bounded by number of elements:

```
int[2]
int[foo+bar]
```

- Bounded by size:

```
int[8#B]
```

The language - Struct Types

- Simple struct:

```
deftype Packet =
  struct
  {
    byte magic;
    uint<32> data_length;
    byte[data_length] data;
  }
```

- Struct with arguments:

```
deftype elf_group =
  struct (elf_off num_idxs)
  {
    elf_group_flags flags;
    elf32_word[num_idxs] shidx;
  };
```

The language - Struct Types

- Field labels:

```
deftype Packet =
  struct
  {
    byte magic;
    uint<32> data_length;
    offset<int,B> data_offset;

    byte[data_length] data @ data_offset;
  }
```

- Pinned structs:

```
pinned struct
{
  uint32 st_info;
  struct
  {
    elf_sym_binding<uint<28>> st_bind;
    elf_st_type<uint<4>> (mach) st_type;
  };
}
```

The language - Struct Types

- Constraints:

```
struct
{
  byte[4] ei_mag : ei_mag[0] == 0x7fUB
                   && ei_mag[1] == ’E’
                   && ei_mag[2] == ’L’
                   && ei_mag[3] == ’F’;
  byte ei_class;
  byte ei_data;
  byte ei_version;
  byte ei_osabi;
  byte ei_abiversion;
  byte[6] ei_pad;
  offset<byte,B> ei_nident;
} e_ident;
```

The language - Union Types

```
deftype Id3v2_Frame =
  struct
  {
    char id[4] : id[0] != 0;
    uint32 size;
    ...
    union
     {
       /* Frame contains text related data.  */
       union
       {
         struct
         {
           char id_asciiz_str = 0;
           char[size - 1] frame_data;
         } : size > 1;

         char[size] frame_data;
       } : id[0] == 'T';

       /* Frame contains other data.  */
       char[size] frame_data;
     };
  };
```

The language - Polymorphic types

- **any**, **any[]**
- Poor man's type polymorphism:
    - everything coerces to any.
    - any coerces to nothing.
- Eventually will transition into **gradual typing**, in a backwards-compatible way:

```
defun efficient_signed
       = (int<32> a, int<32> b) int<32>: { ... }
defun efficient_unsigned
       = (int<32> a, int<32> b) int<32>: { ... }

defun flexible
       = (int<32> a, int<32> b) xint<32>: {...}
defun more_flexible
       = (int<*> a, int<*> b) xint<*>: {...}

defun inefficient = (any a, any b) any: {...}
```

The language - Variables

Block oriented. Lexically scoped.

```
defvar a = 10
defvar b = [1,2,3]
defvar c = { foo = 10, bar = 20L }
```

The language - Mapping

A central concept in poke:

- Poke variables are in memory.
- The IO space is the data being edited (file, memory, ...)
- Both can be manipulated **in the same way**.
- ... or that's the idea.

The language - Mapping

TYPE @ OFFSET `-> MAPPED_VALUE`

- Simple types

```
(poke) defvar a = 10
(poke) defvar b = int @ 0#B
```

- Arrays

```
(poke) defvar a = [1,2,3]
(poke) defvar b = int[3] @ 0#B
```

- Structs

```
(poke) defvar a = Packet { i = 10, j = 20 }
(poke) defvar b = Packet @ 0#B
```

The language - Functions

```
defun ctf_section = (Elf64_Ehdr ehdr) Elf64_Shdr:
{
 for (s in Elf64_Shdr[ehdr.e_shnum] @ ehdr.e_shoff)
   if (elf_string (ehdr, s.sh_name) == ".ctf")
     return s;

 raise E_generic;
}
```

The language - Functions

Optional arguments

```
defun elf_string = (Elf64_Ehdr ehdr, offset<Elf_Word,B> offset,
                    Elf_Half strtab = ehdr.e_shstrndx) string:
{
 defvar shdr = Elf64_Shdr[ehdr.e_shnum] @ ehdr.e_shoff;
 return string @ (shdr[strtab].sh_offset + offset);
}
```

The language - Functions

Variable length argument list. Last argument is an array of **any**s.

```
defun format = (string fmt, args...) string:
  {
   ...
   if (fmt[fi + 1] == 'x')
      res = res + tohex (args[narg] as uint<64>);
   ...
  }
```

The language - Functions

Algol68ism: parameterless functions are homoiconic to variables

```
(poke) defun beast = int: { return 666; }

(poke) beast() + 1
667
(poke) beast + 1
667
```

Architecture

```
+-----------+
| compiler  |
+-----------+      +-------+
     |             |       |
     v             |       |
+-----------+      |       |
|   PVM     | <--->|  IO   |
+-----------+      |       |
     ^             |       |
     |             |       |
     v             +-------+
+-----------+
| command   |
+-----------+
```

The PKL compiler

```
    /--------\
    | source |
    \---+----/
        |
        v
+------------------+
|      Parser      |
+------------------+
|   analysis and   |
|  transformation  |
|      phases      |
+------------------+
|  code generation |
|      phase       |
+------------------+
|  Macro assembler |
+------------------+
        |
        v
   /---------\
   | program |
   \---------/
```

```
(poke) defvar foo = 3
(poke) .vm dis e foo + 10
        note    "#begin prologue"
        canary
        push    0#b
        popr    %r0
        push    0
        pushe   $L15
        note    "#end prologue"
        pushvar 0x0, 0x1a
        push    10
        addi
        nip2
        note    "#begin epilogue"
        pope
        push    0
        exit
$L15:
        pushvar 0x0, 0xd
        call
$L17:
        push    1
        exit
        note    "#end epilogue"
        exitvm
```

The PKL compiler - Passes and phases

```
  [parser]
  --- Front-end pass
  trans1      Transformation phase 1.
  anal1       Analysis phase 1.
  typify1     Type analysis and transformation 1.
  promo       Operand promotion phase.
  trans2      Transformation phase 2.
* fold        Constant folding.
  typify2     Type analysis and transformation 2.
  trans3      Transformation phase 3.
  anal2       Analysis phase 2.
  --- Middle-end pass
  trans4      Transformation phase 4.
  --- Back-end pass
  analf       Analysis final phase.
  gen         Code generation.
```

The PKL compiler - The macro assembler

- Used by the PKL code generator.
- Supports macro-instructions.

```
jitter_label label1 = pkl_asm_fresh_label (pasm);
jitter_label label2 = pkl_asm_fresh_label (pasm);

pkl_asm_insn (pasm, PKL_INSN_OVER);
pkl_asm_insn (pasm, PKL_INSN_OVER);

pkl_asm_label (pasm, label1);

pkl_asm_insn (pasm, PKL_INSN_BZ, label2);
pkl_asm_insn (pasm, PKL_INSN_MOD, ast_type);
pkl_asm_insn (pasm, PKL_INSN_ROT);
pkl_asm_insn (pasm, PKL_INSN_DROP);
pkl_asm_insn (pasm, PKL_INSN_BA, label1);

pkl_asm_label (pasm, label2);

pkl_asm_insn (pasm, PKL_INSN_DROP);
```

The PKL compiler - RAS

Allows to write PVM assembly in a sane(r) way..

```
        .macro gcd @type
        ;; Iterative Euclid's Algorithm.
        over                        ; A B A
        over                        ; A B A B
.loop:
        bz @type, .endloop         ; ... A B
        mod @type                  ; ... A B A%B
        rot                        ; ... B A%B A
        drop                       ; ... B A%B
        ba .loop
.endloop:
        drop                       ; A B GCD
        .end
```

The Poke Virtual Machine

- Stack machine.
- Uses Luca's jitter (`http://ageinghacker.net/jitter`)
- Instruction set: see `src/pkl-insn.def`

The IO Subsystem

```
       "IO spaces"                   "IO devices"

   Space of IO objects <=======> Space of bytes

                                +------+
                     +----->| File |
                     |          +------+
     +-------+       |
     |  IO   |       |
     | space |<------+          +---------+
     |       |       +----->| Process |
     +-------+       |          +---------+

                     :               :

                     |          +-------------+
                     +----->| File system |
                                +-------------+
```

Cache, Transactions, IO update callbacks, ...

Hacking poke - Commands

- Dialectic: DSL vs. command language.
- Need for the later avoided, using a syntax trick:

```
defun foo = (int a, int b = 30, int c) void: { ... }
...
foo (10, 20, 40);
...
foo :c 10 :a 20
...
```

Hacking poke - Commands

```
defun dump = (off64 from = pk_dump_offset,
              off64 size = pk_dump_size,
              off64 group_by = pk_dump_group_by,
              int ruler = pk_dump_ruler,
              int ascii = pk_dump_ascii) void:
{
  ...
}

(poke) dump :from 0xff#B :size 28#B
```

Hacking poke - pickles

- **Collections** of related types, variables, functions.
- File formats: ELF, DWARF, id3v2, ...
- Domains: searching, disassemblers, network packages, ...

Hacking poke - elf.pk

```
deftype Elf_Half = uint<16>;
deftype Elf_Word = uint<32>;
deftype Elf64_Xword = uint<64>;
...
defvar SHT_STRTAB = 3;
defvar SHT_RELA = 4;
...
deftype Elf64_Rela =
  struct
  {
    offset<Elf64_Addr,B> r_offset;
    Elf64_Xword r_info;
    Elf64_Sxword r_addend;
  };
...
defun elf_string = (Elf64_Ehdr ehdr, offset<Elf_Word,B> offset,
                    Elf_Half strtab = ehdr.e_shstrndx) string:
{
 defvar shdr = Elf64_Shdr[ehdr.e_shnum] @ ehdr.e_shoff;
 return string @ (shdr[strtab].sh_offset + offset);
}
```

Testing

```
$ make check
...
Running testsuite/poke.cmd/cmd.exp ...
Running testsuite/poke.map/map.exp ...
Running testsuite/poke.pkl/pkl.exp ...
Running testsuite/poke.std/std.exp ...
exit

                === poke Summary ===

# of expected passes            1147
```

What works

- Basic language: variables, closures, types, etc.
- Mapping.
- Arrays.
- Structs.
- Only one kind of IO device: files.
- `dump` command.

Work in progress

Before first release...

- Struct constructors
- More control sentences.
- Pattern matching
- Commands: search, shuffle, etc.
- Support for unions.
- Support for sets (enums, bitmasks).
- Finish the IO space implementation.
- More IO devices: process, etc.

Future work

... after first release.

- Gradual typing.
- Support for sets (enums, bitmasks).
- Organize pickles better: module system, namespaces.
- Wide strings: `L"foo"`
- Other language improvements.

Project Resources

- Homepage: **http://www.jemarch.net/poke.html**
- Savannah: **http://savannah.nongnu.org/p/poke**
- Mailing list: **poke-devel@nongnu.org**
- IRC channel: #**poke** in **irc.freenode.net**

Will change to www.gnu.org soon.

Hack with me!

See file **HACKING** in the source tree.