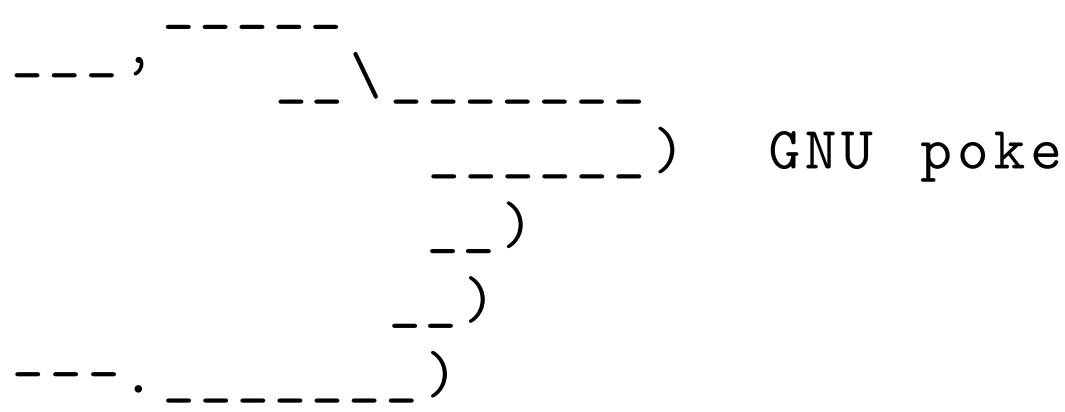


The extensible editor
for structured binary data

Jose E. Marchesi

jemarch@gnu.org

Pass The Salt 2022

Contents

Motivation

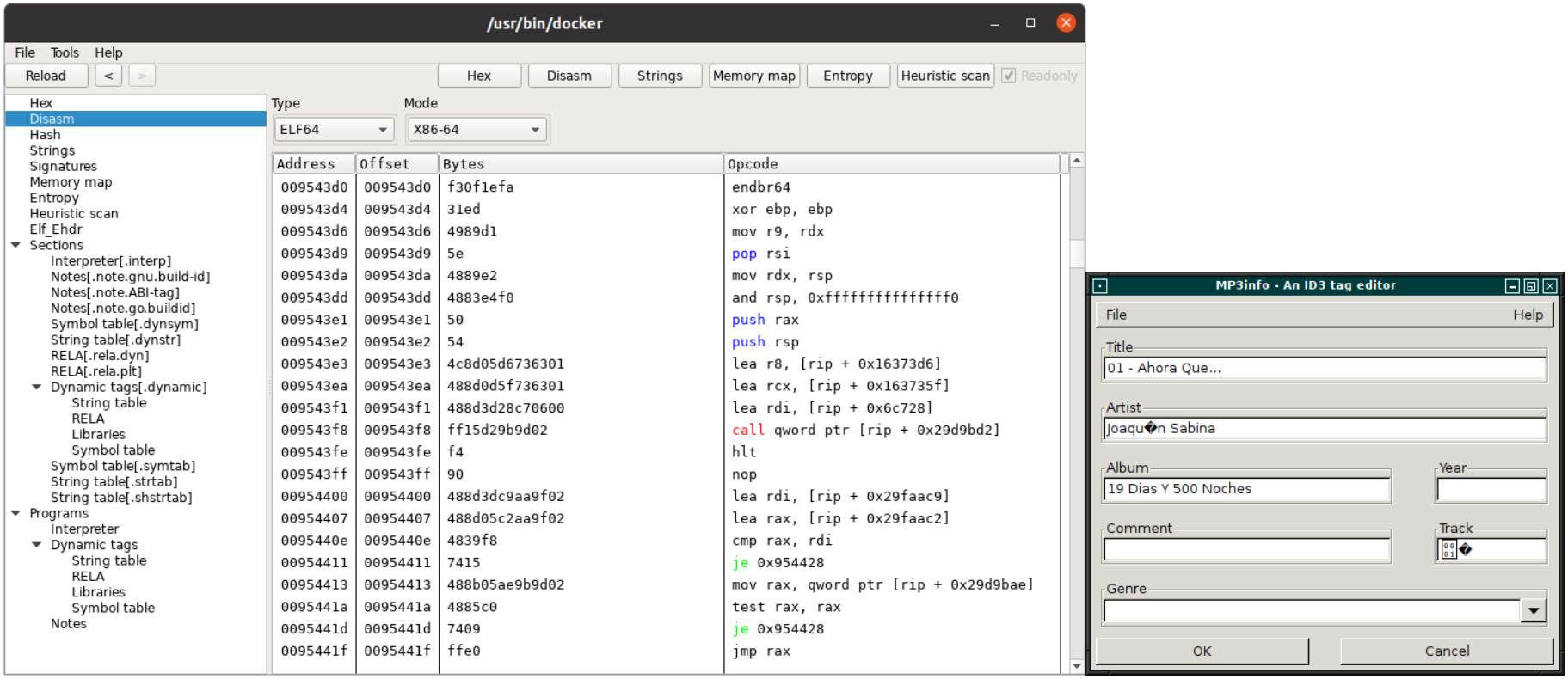
Hmmm gotta to...

```
# 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)
```

Simple Binary Editors

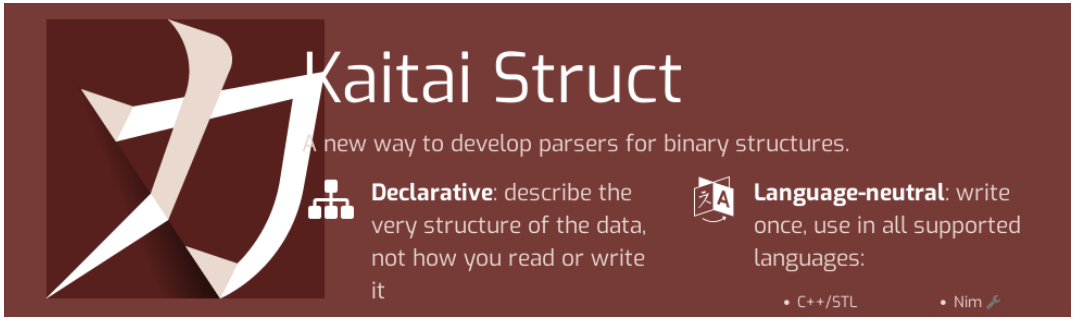
- Cute, easy to use, useful.
- Interactive, **immediate** editing.
- Bytes, strings, sometimes bits.
- Byte dump and ASCII views.
- Fixed capabilities: search, byte diff/compare, entropy, etc.

Specialized Binary Editors



- Pre-defined and fixed data structures: ELF editor, MP3 editor, foo editor, ...
- Not extensible.
- Choke on incorrect data.
- They implement everything but not what **you** need (doh!)

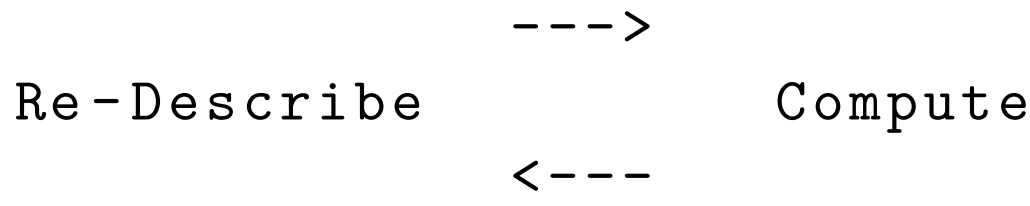
Encoders/Decoders Generators



`Decode ---> Compute [---> Encode]`

- User-defined complex structures.
- XML, JSON, ... “declarative”
- Generate code in several target languages: C, Java, ...
- Non interactive.
- Can’t deal with incorrect data.
- Either bit-oriented (uncommon) or byte-oriented.
- Often no encoders.

GNU poke



- Interactive, **immediate** editing with data integrity.
- User-defined complex structures.
- Powerful and to the point DSL: **Poke**.
  - Interactive, **statically** typed, garbage collected
  - **pickle**: file containing definitions of types, variables, functions, etc, that conceptually apply to some definite domain.
  - No bit-oriented, no byte-oriented: arbitrary **units**
- bit-addressable IO spaces.
- Can edit incorrect/guessed data (non-strict mapping)
- Extensible: write your own editor, commands and tools.

...but poke is not just about editing

- **Precise** and functional documentation of formats/protocols.
- Progressive investigation of a format/protocol, reverse engineering.
- Prototyping.
- Binary utilities (`#!/usr/bin/env poke -L`).
- Filters.
- Integration in other programs (e.g. GDB).

IO Spaces

- Abstract **bit-addressable** space of IO objects:
  - ints - signed integers from 1 to 64 bits wide
  - uints - unsigned integers from 1 to 64 bits wide
  - strings - sequences of NULL-terminated bytes
- Backed by some **byte-addressable** IO device:
  - Files (ios-dev-file)
  - Memory buffers (ios-dev-mem)
  - NBD servers (ios-dev-nbd)
  - Memory of a running process (ios-dev-proc)
  - Standard input/output/error (ios-dev-stream)
  - Zero (ios-dev-zero)
  - Sub IO space (ios-dev-sub)
  - Client application provided IO space (like GDB inferior memory.)

Bytes

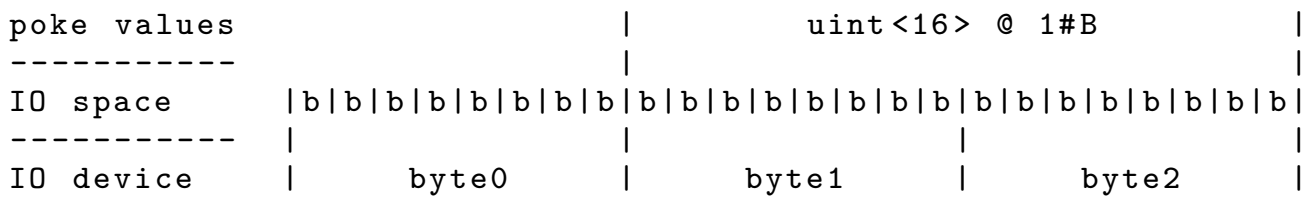
- Data stored in modern computers is fundamentally a sequence of entities called “bytes”:

```
+--------+--------+--------+ ... +--------+
| byte 0 | byte 1 | byte 2 |     | byte N |
+--------+--------+--------+ ... +--------+
```

- Each byte stores a little unsigned integer in the range `0..255`
- In Poke parlance, that is an `uint<8>`, aka `byte`
- `byte @ 4#B`
- `uint<8> @ 4#B = 66`

From Bytes to Integers

- `uint<16> @ 1#B`



- From 1 to 64 bits, signed and unsigned.
- Bit ordering is always big-endian.
- Byte ordering may be either big-endian or little-endian.

Weird Integers - Incomplete Bytes

- `uint<12> @ 0#B`

```
poke values  |        uint<12>       |
-----------  |                       |
IO space     |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----------  |               |               |               |
IO device    |     byte0     |     byte1     |     byte2     |
```

- But what is “the first half of the second byte”?

```
|                  uint<12>                   |
+-------------------------+-------------+-------------+
| a7 a6 a5 a4 a3 a2 a1 a0   b7 b6 b5 b4 :               |
+-------------------------+-------------+-------------+
|           byte 0          |             byte 1            |
```

- In big-endian: `a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4`
- In little-endian: `b7 b6 b5 b4 a7 a6 a5 a4 a3 a2 a1 a0`

Weird Integers - Quantum Bytenics

- `uint<5> @ 0#B`

```
poke values  | uint<5> |
-----------  |         |
IO space     |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----------  |               |               |               |
IO device    |     byte0     |     byte1     |     byte2     |
```

- Let's view the byte as a sequence of bits:

```
|     uint<5>      |
+------------------+-----------+
|  b7  b6  b5  b4  b3  |            |
+------------------+-----------+
|              byte0              |
```

- The value is `b7 b6 b5 b4 b3` regardless of byte endianness.

Unaligned Integers

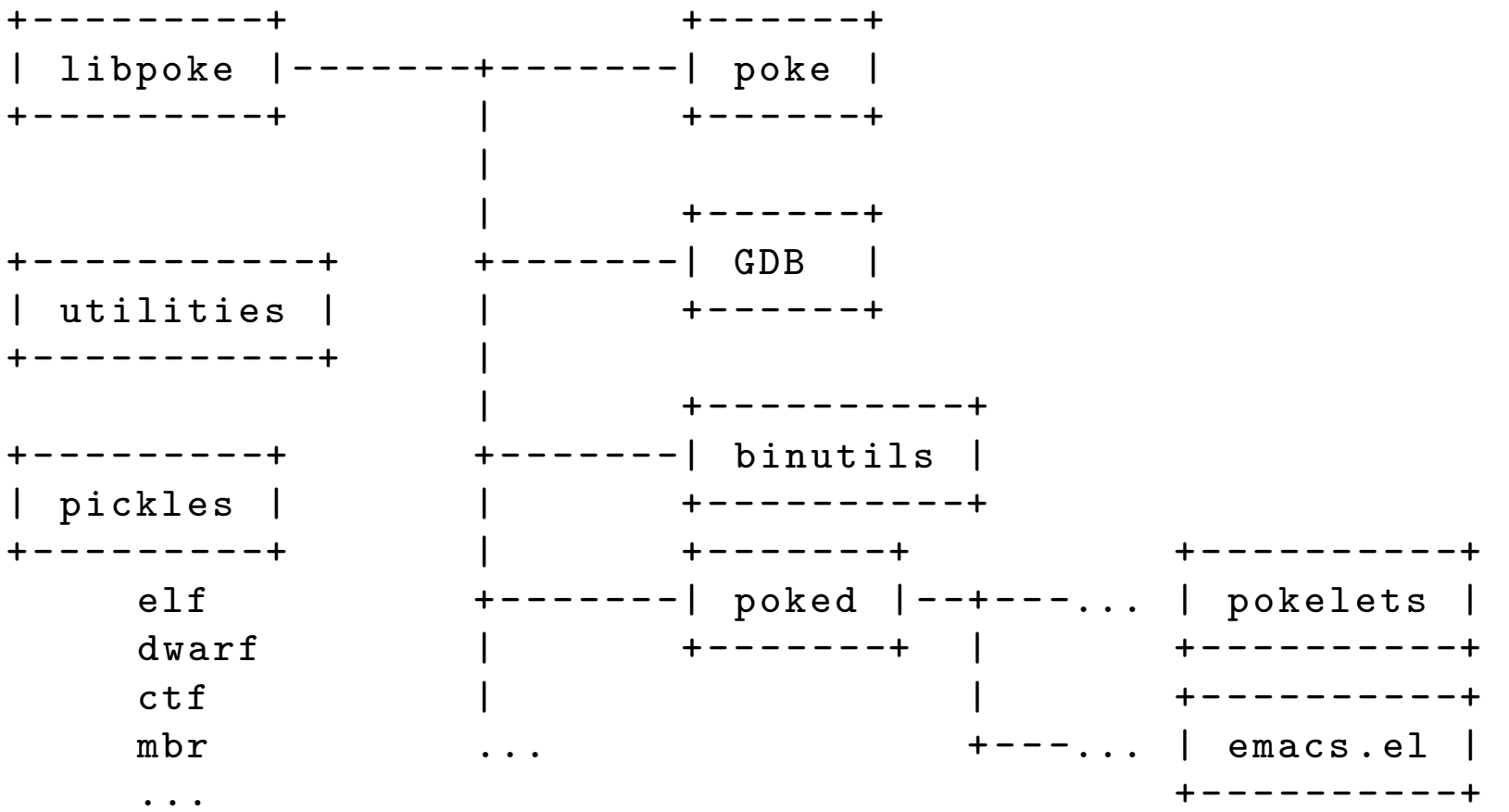
- `uint<16> @ 2#b`

```
poke values       |         uint<16> @ 2#b          |
-----------       |                                 |
IO space      |0|1|1|1|1|1|1|1|0|1|0|0|0|1|0|1|0|1|0|0|1|1|0|0|
-----------   |               |               |               |
IO device     |     0x7f      |      0x45     |      0x4c     |
```

- This helps to visualize it:

```
poke values        |         uint<16> @  2#b         |
-----------        |                                 |
Virtual bytes      | virt. byte1   |  virt. byte2  |
-----------        |               |               |
IO space       |0|1|1|1|1|1|1|1|0|1|0|0|0|1|0|1|0|1|0|0|1|1|0|0|
-----------    |               |               |               |
IO device      |     0x7f      |      0x45     |      0x4c     |
```

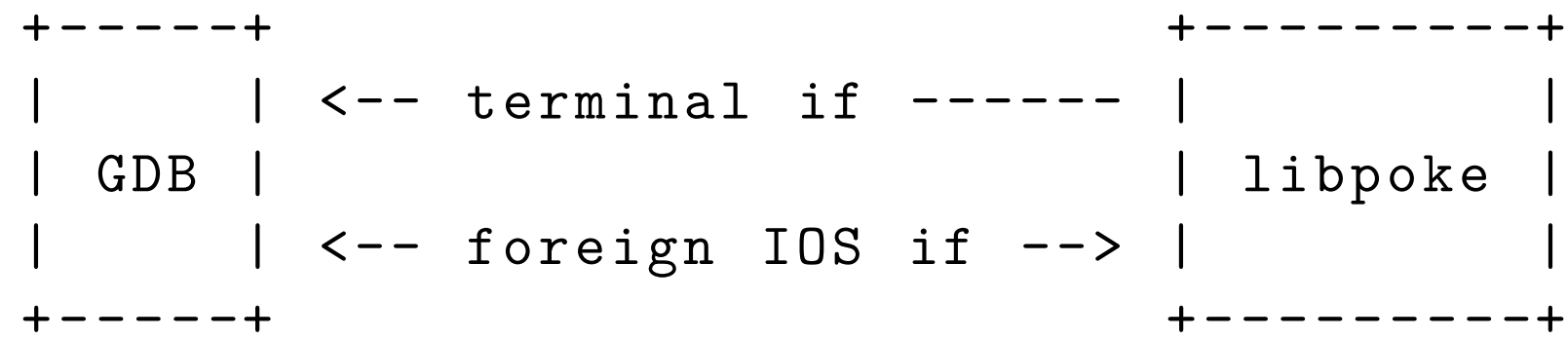
The pokesphere



Segmentation fault time
(demo)

"If the **mountain** will not come to **poke**
poke will go to the **mountain**"

GDB meets poke



- GDB is good at debugging processes
- poke is good at poking at data
- ... guess what, GDB+poke is good at both!

The GNU assembler meets poke (WIP)

- Current data directives

```
.section .text
.set softfloat
.ascii "PS-X EXE"
.byte 0, 0, 0, 0, 0, 0, 0, 0
.word main
.word 0
.word 0x80010000
.word image_size
.word 0,0,0,0
.word 0x8001FFF0
.word 0,0,0,0,0,0
.ascii "Sony Computer Entertainment Inc. for zid"
```

- New `.poke` directive

```
.poke load psxexe
.poke var s = "Sony Computer"
.poke PS_X_EXE { start = $main, size = $image_size, vendor = s }
```

The Project

- Licensed under GPLv3+, both applications and libraries
- Homepage: **https://www.jemarch.net/poke**
- Savannah: **https://savannah.gnu.org/p/poke**
- Mailing list: **poke-devel@gnu.org**
- IRC channel: **#poke** in **irc.libera.chat**
- Pokology: **https://pokology.org**

Join the fun!

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

- Current released poke version is 2.3
- poke 2.x is in maintenance mode
- poke 3.x is under development in the `master` branch.
- The poke daemon, TUIs, GUIs.
- poking at the kernel.