Genspio: Generate Your POSIX Shell Garbage

Sebastien Mondet (@smondet)
OCaml 2017 Workshop, Sep 8, 2017.

Context

Seb: Software Engineering / Dev Ops at the Hammer Lab.

We are a team of software developers and data scientists working to understand and improve how the immune system battles cancer.

More Classical Now

Computational Cancer Immunotherapy

- Run big computational pipelines.
  - Servers with WebUIs, databases.
  - HPC scheduling (Torque, YARN, Google Cloud, AWS, …).
- Deal with precious human data.
- HDFS, (broken) disks, S3, Gcloud Buckets, NFSs.
- Interactive exploration.
  - Direct access for the users (IPython, R, `awk | wc`, …).

Infrastructure

- Need to setup local/cloud/datacenter-ish infrastructure for the lab.
- It's nobody's job.
- Nothing seems there for the “long term.”

→ Make composable tools that allow people to setup/monitor/clean-up their own infrastructure.
  (and it’s more fun, and a better use of software people’s time)

Unix.execv

It always looks simple at first …

```
unix.exeuc "/usr/bin/apt-get" "install" "postgresql"
```

```
let cmd = "install" "postgresql"
in unix.exeuc "/usr/bin/ssh" cmd
```

Who failed? ssh or apt-get?
Ketrew's SSH Call

:facepalm: after :facepalm:

DevOps 101: Install The Oracle JDK

Everybody ends up reading some Stack-overflow answer
Typed/Functional Step Back

1. Start writing simple combinators.
2. Add more typing info.
3. Hit portability / representation problems.
4. Go full-blown EDSL that compiles to pure POSIX shell.

Genspio 0.0.0

- Simple, typed EDSL
- Language.t is a 30+ entry GADT.
  - Boolean, Integer arithmetic + to_string/of_string + (very) basic lists.
  - if-then-else, loops.
  - Redirects, pipes, and captures.
  - Basic exception-like jumping.
- Compiler to POSIX shell.
- Either one-liners, or multi-line scripts.
- Unreadable output by default, but tries to do better when it stat-

Examples

let username_trimmed : string t =
(* The usual shell-pipe operator is |>,
 output_as_string takes stdout from a unit t as a string t ->
| exec `[whoami]` |exec `[tr '-d' ';' '\n']` | output_as_string

Now Jump!

with_failwith (fun error_function ->
let get_user = "the contents of USER: '" get (string "USER") in
(* The operator ->B is string t equality, it returns a bool t that
we can use with if_seq: *)
if_seq

get_user -> username_trimmed
|e| (* 'USER' is different from 'whoami', system is broken, so
  we exit using the failwith function: *)
error_function

message:[string "I'm dying"] -> return[int :]
)

CLI Parsing

let cli_spec =
| Command_line_Arg {
  string ~doc:"The URL to the stuff" [-u]; "-url"]
-default:() value
  & flag [-u]; "--all-in-temp" & doc:"Do everything in the temp-dir"
    & string [-t]; "--temp-dir" & doc:"Override the downloaded file-name"
    & default:() value
  & string [-t]; "--temp-dir" & doc:"Use <dir> as temp-dir"
    & default:() value
  & usage]

Command_line.parse cli_spec

Line-by-line

let on_stdin_lines -body =
let fresh =

```
sprintf "var_%d_%s" Random.int 10 999
(Genspio.Language.to_one_liner (body [string "bouh"]))
|> Digest.string |> Digest.to_hex
```

Nice Call

```
(' ... ')
ex

factory ["ARGS"; exe]
|> exec ["awk"; "if ([ spooky ] { print $3 } elseif ( print $1 )}
|> on_stdin_lines begin fun line ->

seq {
  call [string "printf"; string "line %n\n"; line];
  call [string "cp"; line; string ["/tmp" // basename]];
}
```

Under The Hood: String Representation

That's when "crazy" really means "insane."

```
| Output_as_string ->
sprintf "$ARGS ->

od -t c1 -An -v | tr -d " \n" | (continue e)
```

Vs

```
let expand_octal s =
sprintf [sh] printf -- "$ARGS ->
```

Still Work To Do

```
let to_argument varprefix =
let argument =\n  argument \n    \n    |~body: | doc:"\n    \n    function |

| String = \n  argument [\n    argument [\n      | usage "\n      argument [\n        | usage [\n```

Future work: 2 string types …

C-Strings Vs Byte-arrays

In the beginning there was UNIX …

```
in
```

Testing, Locally

Test tries all the shells it knows about on the current host:
We get the usual report:

```
tmpfs 512.0K 0 512.0K 0% /dev
/dev/root 46.5M 2.9M 42.7M 6% /
```

```
Filesystem Size Used Available Use% Mounted on
```

-root@OpenWrt:/# df -h

---

Example of Rabbit Hole

For a given shell, trying:

```
$shell c 'exec 4043 ; echo 'Exec-returns: $?'' ; echo "Shell-returns: $?"
```

The POSIX ones:

- `shell=dash, shell=sh, shell='busbox ash'` Shell-returns: 2
- `shell=ksh, shell='ksh'` Shell-returns: 1

The non-POSIX ones:

- `shell=bash, shell=sh` Exec-returns: 1 Shell-returns: 0

→ even bash not always POSIX.

Seccotree

### Real-world example.

- Library of Hammerlab-like deployment lego-bricks.
  - Kretrew, Coclobas, NGinx auth, TLS tunnel
- Let's Encrypt, GCloud DNS, …
- "Interactive exploration containers."
- Kubernetes/AWS-Batch clusters.
- Take down everything, restart partially …
- With pre-assembled (but configurable) "examples" for GCloud, AWS, and "Local-docker" standard setups.

https://github.com/hammerlab/seccotre

### Secotrec

- Got to "scale" Genpio:
  - Quickly hitting max length of command line argument.
  - "Standard Library" that may merge into Genspio.
  - Integration with docker-compose.
- "GCPocalypse":
  - Too easy for users to setup their own infrastructure.
  - Forgetful about cleaning up.
  - → our benefactor said it's too much
  - Fast move of all ops back to local infrastructure.

Habust

Simple "build-stuff" EDSL, compiled to a `Makefile` + scripts:

- Download Qemu images.
- Setup/start qemu VM.
- Run recipe on the VM in a mostly restartable way.
- Ketrew, Coclobas, NGinx auth, TLS tunnel
- Add artifacts from the VM into a .tgz (e.g. an executable + output of `lsmod`)

```
habust
```

```
https://github.com/smordet/habust
```

```
Example of Rabbit Hole
```

```
For a given shell, trying:
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Ketrew on ARM64

Silence on ARM64

Could not get the graphical apps I wanted to show:

Future Work

• Byte-array Vs C-String type.
• GADT Vs TTFI discussion (cf. this afternoon): we want to call the compiler within a “script” to use its output as a literal string
• More combinators (integration of Secotrec/Habust functions).

The End

Questions?