OCaml Under the Hood:

SMARTPY

OCaml Workshop 2020-08-28

Seb Mondet, TQ Tezos

The Who

- Software Engineer at TQ Tezos → Improve Tezos Ecosystem
  - Core `tezos/tezos` development.
  - Smart Contracts.
  - Helping partners get up to speed and build on Tezos.
  - Standardization efforts.
  - Contribute to tooling, incl. SmartPy.
- I'm not the main developer of SmartPy, cf. also
  - François Maurel → Main architect.
  - Roland Zumkeller → Compiler + Decompiler + ...
  - Rodrigo Quelhas → Infrastructure + WebUI.

Tezos

Tezos is a *cryptocurrency* usually well-known in the OCaml world:

- Proof of Stake — Baking
- On-Chain Governance — Self Amending Protocol
- Smart Contracts
- OCaml implementation

Michelson

The "VM / scripting language" of Tezos.

- Programs that the whole network run & validate.
- Stack-based & functional core.
- Clear semantics and typing rules.
- Emphasis on formal methods.

Writing smart-contracts is HARD ...

- Bugs can be very expensive (money locked or stolen),
- VM-languages like Michelson are very low-level.

SmartPy

Python **library** for writing smart-contracts on Tezos

- Generate Michelson + Test Contracts
- Tooling:
  - WebIDE and CLI tools,
  - Simulate & analyze,
  - Deploy & interact.
- Big mix of OCaml, Python, and Javascript.

Why Python

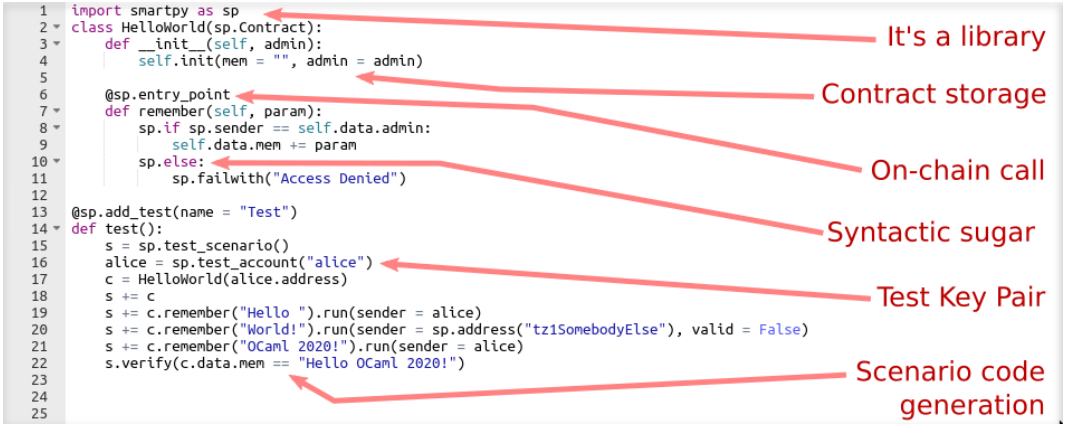
- One of the most popular languages in the world,
- Intuitive syntax,
- Good meta-programming capabilities,
- New users *believe* they already know it
  $\rightarrow$ Tezos Gateway Drug :)

Python script to Simulation/Michelson

SmartPy programs generate **SmartML** expressions

- SmartML is an imperative, type-inferred intermediate representation
- SmartML is implemented in an **OCaml** library:
  - Compiled to Native (tests, CLI tools) and to Javascript (WebIDE, end-user CLI application),
  - Type inferrence, Program Analyses,
  - Interpreter, incl. **test scenarios** language,
  - Compiler to Michelson (with many Michelson to Michelson optimizations).
  - In progress: a decompiler.

Example 0: Full Contract



WebIDE: Demo

A.k.a. `webide-demo.mp4`.

WebIDE: Implementation

What happens:

- Python code executed with the Brython interpreter.
- Constructs SmartML Programs.
- Contract enters the `js_of_ocaml` world:
  - Type inference / checking,
  - Simulation
  - Compilation
  - Back to the UI to construct the HTML “right pane”

SmartPy-CLI

```
# Install on any Unix with `npm` & `python` (3):
sh <(curl -s https://smartpy.io/dev-202007…/cli/SmartPy.sh) local-install-auto

# Compile and run tests:
~/smartpy-cli/SmartPy.sh test <myscript.py> <output-directory>

# Just compile a given contract within a python script:
~/smartpy-cli/SmartPy.sh compile welcome.py "Welcome(12,123)" /tmp/welcome
```

“Portable” OCaml CLI App

`SmartPy.sh` is a `bash` script ...

- Knows how to install the `smartpy-cli` distribution: python and JS files + `npm` dependencies,
- and call the main OCaml application *if available*:

`smartml-cli.js` is the JS “main”

- Concatenation of a `prelude.js` that loads `npm` packages,
- *and* the result of `js_of_ocaml`.

A bit slow (esp. startup), but does the job.

Spice

```
(rule
  (targets smartml-cli.js)
  (deps node_main.bc.js prelude.js)
  (action
    (with-stdout-to smartml-cli.js
      (progn
         (run cat prelude.js)
         (run sed
          "s@joo_global_object.console.log(cmd);@// removed console.log@ ; \
               s/.execSync(cmd/.execSync(cmd,{stdio: 'inherit'}/"
               node_main.bc.js)
         (echo "}) ()\n")
      ))))
```

(that sed is fixed upstream → ocsigen/js_of_ocaml#979)

Pre-hack Dot JS

```
// ...
const library = {
  bs58check: require('bs58check'),
  sodium: require('libsodium-wrappers-sumo'),
};
// ...
(async() => {
  await library.sodium.ready;
  await library.bs58check.ready;
  global.sodium = library.sodium;
// __INSERT_HERE__
```

Libsodium & Friends

```
(* ... *)
module Ed25519 : sig
  val verify_signature :
       Crypto_bytes.t
    -> message:Crypto_bytes.t
    -> public_key:Crypto_bytes.t
    -> bool
    [@@js.global "sodium.crypto_sign_verify_detached"]

  val sign :
    message:Crypto_bytes.t -> secret_key:Crypto_bytes.t -> Crypto_bytes.t
    [@@js.global "sodium.crypto_sign_detached"]
(* ... *)
```

Some Bit Flipping

```
let binary_string buf b =
  let lgth = String.length b in
  Buffer.add_char buf ((lgth lsr 24) land 0xFF |> Char.chr);
  Buffer.add_char buf ((lgth lsr 16) land 0xFF |> Char.chr);
  Buffer.add_char buf ((lgth lsr 8) land 0xFF |> Char.chr);
  Buffer.add_char buf ((lgth lsr 0) land 0xFF |> Char.chr);
  Buffer.add_string buf b
in
let z_big_int buf bi =
  (* See tezos: src/lib_data_encoding/binary_writer.ml:150 *)
  let open Big_int in
  let original_sign =
    if sign_big_int bi = 1 || sign_big_int bi = 0 then 0 else 0b0100_0000
  in
  let absed = abs_big_int bi in
  let rec pack_bits buf how current_bi =
    let n_bits, ones, sign =
      match how with
      | `First_six -> (6, 0b11_1111, original_sign)
      | `Remaing_chunks -> (7, 0b111_1111, 0)
    in
    let n_little = and_big_int current_bi (big_int_of_int ones) in
    let last_one =
      if eq_big_int n_little current_bi then 0 else 0b1000_0000
    in
```

Error Messages

- Python interpreter
- SmartML Type inference/check
- Interpreter
- Compiler
- Tezos-type-checker/contract-origination

Some Polyglotism Challenges

- Knowledge of **implementations**:
  - Brython Vs regular python interpreter.
  - Browser $\times$ Node.js $\times$ `js_of_ocaml`.
- Slight culture clash on OCaml style Vs More traditional ML Vs Haskell.
- Convincing to everybody switch to OCamlFormat: 16-line `.ocamlformat`!
- Speed & performance of Javascript (CLI & Web)

FA2

Part of standardization "Multi-asset Contract Interface" → One reference implementation.

`gitlab.com/smondet/fa2-smartpy`

Just `multi_asset.py` is 1 KLoC (`FA2.py` in SmartPy's IDE).

Really uses meta-programming: 12 boolean configuration switches

Heavy Benchmarking (*blog post pending ...*).

OCaml code generation from the Michelson output, used to build a mini-wallet/benchmarks command-line application

Success Story

- Good popularity within the Tezos ecosystem,
- Telegram help-channel:
  $>$ 200 members,
- Twitter account $\rightarrow$ about 600 followers.
- There are already 3rd party online courses: blockmatics.io or "Cryptobots vs Aliens", and most hackathons include SmartPy.
- Financial applications such as ChainLink already build on SmartPy,
- Other tools from the ecosystem like ConseilJS natively support SmartPy.

Roadmap / WIP

- Making sandbox testing *more* available to end-users
- Decompilation take
- Other analyses:
  - Abstract Interpretation: ownership, etc.
  - Gas usage prediction.
- Other generation targets:
  - Storage schema / parsing code
  - WhyML, Coq.

The End

Thanks !

- Website, docs, WebIDE: `smartpy.io`
- Slides:
  `wr.mondet.org/slides/SmartPy@OCaml2020/20200828-smondet-smartpy.pdf`
- Me: `seb.mondet.org`
- TQ: `tqtezos.com` ← **We're hiring: OCaml, Haskell, DevOps, WebDev, ...**

Example 1: Tezos Primitives

```
@sp.entryPoint
def setCurrentValue(self, params):
    thingToSign = sp.pack(
        sp.record(
            o = self.data.currentValue,
            n = params.newValue,
            a = sp.self,
            c = self.data.counter))
    sp.verify(
        sp.checkSignature(
            self.data.bossPublicKey, # Only tz1 in browser for now
            params.userSignature,
            thingToSign))
    self.data.currentValue = params.newValue
    self.data.counter = self.data.counter + 1
```

Example 2: Some OO

```
class MultiSigFactory(sp.Contract):
    def __init__(self):
        # ...

    @sp.entryPoint
    def checkSigsAndDo(self, params):
        # ...
        self.onOK(contract)

    def onOK(self, contract):
        pass

class MultiSigFactoryWithPayment(MultiSigFactory):
    def onOK(self, contract):
        sp.send(contract.owner, contract.amount)
```

Example 3: Some Meta-programming

```
class NimGame(sp.Contract):
    def __init__(self, size, bound = None, winnerIsLast = False):
        self.bound        = bound
        self.winnerIsLast = winnerIsLast
        self.init(deck = range(1, size + 1), size = size,
                  nextPlayer = 1, claimed = False, winner = 0)

    @sp.entryPoint
    def remove(self, params):
        # [...]
        sp.verify(params.cell < self.data.size)
        sp.verify(1 <= params.k)
        if self.bound is not None:       # ------>  NOT AN sp.if !
            sp.verify(params.k <= self.bound)
        sp.verify(params.k <= self.data.deck[params.cell])
```

Non-Hello-World Examples

Within the WebIDE:

- Calculator
- Fungible and non-fungible assets
- Multisig contracts
- Escrow contract
- State channels (under development)
- Games: tic-tac-toe, nim, chess

See also on SmartPy.io.