SCaml for Babylon

DaiLambda, Inc.
Jun FURUSE / 古瀬 淳

ReFX, Kyoto, 2019-10-24

SCaml

OCaml to Michelson compiler:

- Strict subset of OCaml
- Small: 2500 loc of `.ml`
- Built over OCaml's `compiler-libs`
- Covers all the opcodes except `CREATE_CONTRACT`

Babylon upgrade for Tezos

Highlights which may interest us:

Address cleaning

`KT1.*` now always has code.
`tz[1-3].*` never has code.

Closures

`LAMBDA` is now not only a code block but with an env.
`APPLY` is for partial application.

Entry points

Easily select the part of codes to be executed.

Opcode for chain ID

`CHAIN_ID`: to prevent replay attacks

Address cleaning

Past

`KT1.*` (KonTract type 1) was for delegation and smart contracts.

Now

Delegation is available for `tz[1-3].*`, too.

`KT1.*` now always with a smart contract.

Once contractless `KT1.*` now has `manager.tz`, and its safety is proven:

https://gitlab.com/nomadic-labs/mi-cho-coq/blob/master/src/contracts/manager.tz

Closures in Michelson

Now `LAMBDA` can carry an environment. No hand encoding of closures is required.

```
APPLY :: α  ;  (α * β) → γ  ;  S
      ⇒ β → γ  :  S

APPLY |  (A : α) ; { code } ; ..
      ⇒ { PUSH α A ; PAIR ; code } : ..
```

Closure generation outline:

```
let x = 1 and y = 2 in
let f = fun z -> x + y + z

LAMBDA { .. /* fun (x,(y,z)) -> x + y + z */ } ;
PUSH 1 int ; APPLY # to apply x
PUSH 2 int ; APPLY # to apply y
PUSH 3 int ; EXEC  # execute f
```

Pre-Babylon: choosing “exec mode”

Use `or` type in parameter and branch by `IF_LEFT`:

```
parameter (or int          # for case 1
               (or string # for case 2
                   bool   # for case 3
           )) ;
code {
  DUP ; CAR ;       # to get parameter
  IF_LEFT {
    .. # for case 1 to handle int
  } { IF_LEFT {
    .. # for case 2 to handle string
  } {
    .. # for case 3 to handle bool
    } }
}
```

Contract call required boring positioning:

```
Right (Left "hello")
```

“Entrypoints” in Babylon

It is just a hack. Tag parameters of modes with `%name` annotations:

```
parameter (or (int          %case1)
              (or (string %case2)
                  (bool   %case3))
          )) ;
code {
  /* The same code.  No fancy things */
}
```

Contract call with a tag: `%case2 "hello"`:

```
PUSH "KT1...." address ;
CONTRACT %case2 string; # Not CONTRACT (or int (or string bool))
PUSH "hello" string ;
TRANSFER_TOKENS ..
```

Opcode for chain ID: `CHAIN_ID`

To prevent **replay attacks**.

- Contracts like `multisig.tz` requires a counter to be signed, otherwise the same signature could be used to replay the operation.
- Tezos test phase forks a test net, which has the same blockchain state, including the counter of `multisig.tz`
- Signature made in the test net can be used to replay the same operation in the mainnet.

Signing also against the **chain identifier**, we can prevent this replay attack between forks.

SCaml updates for Babylon

Native closure

No more typing hack to track free variables!

Entry points

Allow multiple entries by `let [@entry] x = ..`

Chain ID support

Added `Global.get_chain_id : unit -> chain_id`

Native closure

Hours of work for row-type variables are removed ;-)

Entry points

Multiple toplevel declaraitons with `@entry` generate the branching:

```
let [@entry] case1 (n : int)    storage = ...
let [@entry] case2 (s : string) storage = ...
let [@entry] case3 (b : bool)   storage = ...
```

```
parameter (or (or (int    %case1)
                  (string %case2))
              (bool %case3)) ;
storage .. ;
code {
  DUP ; CAR ;
  IF_LEFT { IF_LEFT { .. /* case1 */ }
                    { .. /* case2 */ }
            .. /* case3 */ }
}
```

Chain ID

Easy.

Findings

Michelson values of `chain_id` are not comparble.
Only the use of it is to `PACK` with other data (then sign).

Optimizations

Program transformation in `IML` level:

- `(fun x → e1) e2` ⟹ `let x = e2 in e1`
- `let x = e2 in e1` ⟹ `e1[e2/x]`, if `x` appears only once.

Purity makes these transf. very trivial.

Todo: `CREATE_CONTRACRT`

Roughly:

```
val create_contract :
  ('param -> 'stroage -> operation list * 'storage)
  -> tz
  -> 'storage
  -> operation * address
```

Problems:

- The code must be obtained from the function, **at complie time**.
- How to handle multiple entry points? Seems almost impossible to do this within a subset of OCaml.

Source code of SCaml for Babylon

https://gitlab.com/dailambda/scaml/tree/babylon

Yell me if you have no access rights!