Crotalinae

Metaprogramming for Ethereum Smart Contracts
expressed in Scala's Type System

Increase Dev Adoption

- Unlock Smart Contracts **adoption with a JVM**-based language: code them *from* Scala or *in* Scala

Wonderful Benefits

- Smart Contracts code generator is itself **a strictly typed program**
- Write a Contract using structs and definitions of **Crotalinae DSL**, and if it compiles, you're **safe**
- **Export** Smart Contract in *Vyper* as a single plaintext and check it visually if needed
- *(WIP)* Code directly in Scala: **Scala source code** is translated to Crotalinae DSL using macros

Demonstration

```
val f = `@public` @:
  sumArgs.funcDef( name = "sumSome", uint256) { args ⇒
  for {
    c ← 'c :=: `++`(args.ref('a), args.ref('b))
    d ← 'd :=: `++`(args.ref('b), c)
    _ ← d :=: c
    sum ← `++`(args.ref('a), d).toReturn
  } yield sum
}

println(f.toVyper)
```

```
@public
def sumSome(a: uint256, b: uint256) -> uint256:
  c = a + b
  d = b + c
  d = c
  return a + d
```

How we made it

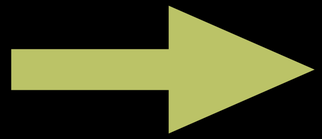
We use only the crazy stuff:

- Heterogenous lists from Shapeless
- Free Monad from Cats
- Natural transformation from DSL to Writer Monad
- DSL constraints are validated on type level

Auction Demo

```
object Auction extends App {
  import Expr.Defs._

  val data = ProductType.self(
    ('beneficiary ->> public(address)) ::
      ('auction_start ->> public(timestamp)) ::
      ('auction_end ->> public(timestamp)) ::
      ('highest_bidder ->> `public`(address)) ::
      ('highest_bid ->> `public`(wei_value)) ::
      ('ended ->> public(bool)) :: HNil
  )

  val beneficiary = data.ref('beneficiary)
  val auction_start = data.ref('auction_start)
  val auction_end = data.ref('auction_end)
  val highest_bid = data.ref('highest_bid)
  val highest_bidder = data.ref('highest_bidder)
  val ended = data.ref('ended)

  val initArgs = ProductType(('_beneficiary ->> address) :: ('_bidding_time ->> timedelta) :: HNil)

  val _beneficiary = initArgs.ref('_beneficiary)
  val _bidding_time = initArgs.ref('_bidding_time)

  val init = `@public` @: initArgs.funcDef(
    name = "__init__",
    Void
  ) { args =>
    for {
      _ <- beneficiary :=: _beneficiary
      _ <- auction_start :=: `block.timestamp`
      _ <- auction_end :=: `+:+`(auction_start, _bidding_time)
    } yield Void
  }

  val bid = `@public` @: `@payable` @: ProductType.hNil.funcDef(
    name = "bid",
    Void
  ) { args =>
    for {
      _ <- `assert`(`<<`(`block.timestamp`, auction_end))
      _ <- `assert`(`>>`(`msg.value`, highest_bid))
      _ <- `if`(`not`(`:===:`(highest_bid, `msg.value`)), {
        () =>
          send(highest_bidder :: highest_bid :: HNil).liftF.map(_ => Void)
      })
      _ <- highest_bidder :=: `msg.sender`
      _ <- highest_bid :=: `msg.value`
    } yield Void
  }

  val end_auction = `@public` @: ProductType.hNil.funcDef(
```

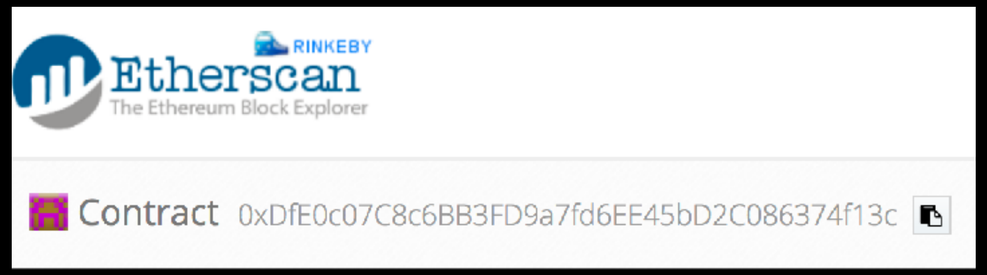
Auction



Source Code | Bytecode ✔ | ABI ✔ | LLL ✔

```
beneficiary: public(address)
auction_start: public(timestamp)
auction_end: public(timestamp)
highest_bidder: public(address)
highest_bid: public(wei_value)
ended: public(bool)

@public
def __init__(_beneficiary: address, _bidding_time: timedelta):
  self.beneficiary = _beneficiary
  self.auction_start = block.timestamp
  self.auction_end = self.auction_start + _bidding_time


@payable
@public
def bid():
  assert block.timestamp < self.auction_end
  assert msg.value > self.highest_bid
  if not self.highest_bid == msg.value:
    send(self.highest_bidder, self.highest_bid)

  self.highest_bidder = msg.sender
  self.highest_bid = msg.value


@public
def end_auction():
  assert block.timestamp >= self.auction_end
  assert not self.ended
  self.ended = True
  send(self.beneficiary, self.highest_bid)
```

Outside My Computer



- Contract written in Scala!
- Deployed on Rinkeby Testnet!

For Tech Dive

Visit github.com/fluencelabs/hackethberlin