Declarative Smart Contracts

Markus Völter
1

Context
Any Turing Complete Program running on a Blockchain.
An actual contract, executed automatically.

Multiple Parties.
Decision ||
Agreement ||
Coordination.

(Legally) Binding &
Trusted.

Formal Language.
Checkable.
Understandable.
„Event Tracking“
Progress over time
Contract Definition → Contract Execution
Understand Behavior
Functional Correctness

Non-Repudiability
Verified Behavior
Non-Gameability

BC
Blockchains can provide certain non-functional properties to executable contracts.

Blockchains are a suitable (partial) implementation technology iff these properties are needed.
Verification

Ensure that the program performs correctly the things the program text tells it to do.

Validation

Ensure that the program does the correct things, wrt. to the requirements.
Verification

Contract Execution

Ensure that the program performs correctly the things the program text tells it to do.

Validation

Contract Definition

Ensure that the program does the correct things, wrt. to the requirements.
Correct-by-Construction

The language/framework/API/modeling tool doesn’t allow a particular class of mistakes.

Analysis-and-Fix

You analyze the code/model after the fact and try to find problems which devs then fix.

DSL

Domain Specific Language
Well, people realize that these contracts have to run reliably, after all, the programs deal with real values now.

Some Blockchain Guy
Not the first community to realize ... 😊
Lots of History & Research

Computational Law

Obligation, Permission
Ordering, Causality, Time
Event, State
Lots of History & Research

Composing contracts: an adventure in financial engineering

POETS Process-oriented event-driven transaction systems

Contracts in Programming and in Enterprise Systems

Domain-Specific Languages for Enterprise Systems
https://bitbucket.org/jespera/poets/raw/c0ee7194ce57d2ad6ca8894c8a44e88e546d5f4a/doc/poets-techreport/tr.pdf
2
Solution
An Architecture For Smart Contracts

MPS / Convecton

DSL Logistics
DSL Finance
DSL ...

Executable Multi-Party Contract Language

KernelF

Iulia
Java
Hyperledger

Ethereum VM/Network

HTML JavaScript CSS

SMTLIB

Z3

Distribution, Trust
Interaction, Integration
Correctness

Generate to verification tools to build more confidence beyond type checking.
Declarative Description

MultiPartyBooleanDecision

A declarative, configurable specification of how a number of parties makes a (Boolean) decision.
A MPBD instance maintains the state of a decision process as it evolves over time.

Here, we play with an instance in the interactive REPL.
Combination with State Machines

More complex contracts are modeled as state machines; events are the API.

```plaintext
event openAccess  // go to the mode where we allow new guys to request to join
event requestAccess(newGuy: party)  // a new guy wants to join the deciders
event terminateAccessRequest(who: party, newGuy: party)  // kill a decision procedure
event voteForAccess(voter: party, newGuy: party)  // vote for a new guy to become decider
event letsSell  // go to the state where we maintain the sell/no-sell decision
event voteForSelling(who: party)  // vote for the sale decision
event voteForStopSelling(who: party)  // vote against the sale decision
```

Internally, the use BPBDs.

<table>
<thead>
<tr>
<th>multi-party-decision</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial parties: bernd, klaus</td>
<td></td>
</tr>
<tr>
<td>dynamic? ✗</td>
<td>sealable? □</td>
</tr>
<tr>
<td>procedure: unanimous</td>
<td>time limit: &lt;none&gt;</td>
</tr>
<tr>
<td>turnout: &lt;none&gt;</td>
<td>revokable? ✗</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>multi-party-decision</th>
<th>AccessControl</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial parties: bernd, klaus</td>
<td></td>
</tr>
<tr>
<td>dynamic? ✗</td>
<td>sealable? □</td>
</tr>
<tr>
<td>procedure: majority</td>
<td>time limit: 20000</td>
</tr>
<tr>
<td>turnout: &lt;none&gt;</td>
<td>revokable? □</td>
</tr>
</tbody>
</table>

```plaintext
var sale = run(Sale)
var pendingAccess = box(map<party, AccessControl>())
observable query currentlySelling = sale.decisionTaken
```
Combintation with State Machines II

Here, a transition action creates a new AccessControl instance ... 

```scala
on requestAccess(newGuy) [=!isDecider/R(newGuy)] : {
  val acc = run(AccessControl)
  pendingAccess.update(it.put(newGuy->acc))
  acc.addParties(sale.registeredParties)
}
```

The state of that instance is then used in guard condition for the top level SM.

```scala
on voteForAccess(voter, newGuy) [isPending/R(newGuy) && isDecider/R(voter)] : {
  val acc = pendingAccess.val[newGuy]
  acc.vote(voter)
  if acc.decisionTaken then {
    sale.addParty(newGuy)
    pendingAccess.update(it.remove(newGuy))
  } else none
}
```
Preventing Game Theoretical Attacks

Only „valid“ senders can enter this state.

```plaintext
state playing [senderIs(players)] {  
on offerBid(money) : bids := bids.put(sender->money)  
  ...  
}
```

Events can only arrive at limited rate.

```plaintext
state requesting [rate(3/1000|commands-only)] {  
  ...  
}
```

States must be entered turn-by-turn.

```plaintext
state playing [senderIs(players)] {  
  state bidding [takeTurns(players|ordered|after 1000 remove)] {  
    on offerBid(money) : bids := bids.put(sender->money)  
    if [timeInState > 2000] -> finished  
  }  
  ...  
}
```
An Architecture For Smart Contracts

IDEA
Example: HyperCSL

Lisp (Clojure) based internal DSL for specification of general commercial contracts.

Inspired by Simon Peyton Jones and Jean-Marc Eber and the POETS group at CPHU and ITU in Denmark.

Uses Ken Adams’ Categories of Contract

Language as fundamental semantic building blocks. Interpreter and UI in prototype stage.
Example: HyperCSL

(con :c2
    (obs :exercised-timely? :p1)
    [(obli :p3
            "lender"
            "borrower"
            (action :payment (event :adv$1000) "Advance loan to borrower."
                    (tw (dt 2014 6 1) (dt 2014 6 2))))

(con :c3
    (obs :fulfilled-timely? :p3)
    [(obli :p4
            "borrower"
            "lender"
            (action :payment (event :pay$550) "Repay first installment."
                    (obs :if (obs :event-occured? :event-of-default)
                          (tw (obs :first-time-of :event-of-default))
                          (tw (dt 2015 6 1))))

(obli :p5
      "borrower"
      "lender"
      (action :payment (event :pay$525) "Repay second installment."
              (obs :if (obs :event-occured? :event-of-default)
                     (tw (obs :first-time-of :event-of-default))
                     (tw (dt 2016 6 1))))])}
Example: HyperCSL

A UI to visualize the interactive execution of CSL contracts.

Loan Agreement

c1 - Condition: event-occurred? - executed - fulfilled

p0 - Preamble

This loan agreement dated 2014-06-01, by and between Lender Bank Co. and Borrower Corp., will set out the terms under which Lender will extend credit in the principal amount of $1,000 to Borrower with an un-compounded interest rate of 5% per annum, included in the specified payment structure.

p1 - Discretion - Exercised: true - at: within

Borrower may, by way of notice loan-request within the required time window of: 06/01/2014

p2 - Policy: not-activated

if:

discretion-late?

  ■ p1

then:

agreement-terminated at ?

c2 - Condition: exercised-timely? - p1 - fulfilled

p3 - Obligation - fulfilled: true - at: within

lender must adv $1000 in favor of borrower within 06/01/2014 - 06/02/2014

p4 - Obligation - fulfilled: true - at: before

borrower must pay $550 in favor of lender within 06/04/2015
Tooling
An Architecture For Smart Contracts

KernelF is an extensible functional language used at the core of DSLs.
DSL Development

GPL Extension
Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL

New Language
Analyze Domain to find Abstractions
Define suitable, new notations.
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

Formalization
Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“
Functional Features

Functional, no state at its core.
Purity + Effect Tracking
The usual types, literals and op’s
Various Conditionals
Functions and Blocks
No null, only opt<T>
Error Handling attempt<T|E-1,... E-n>
   try <e> => <s> error <E-1> => <e-1> ... error <E-n> => <e-n>
Immutable Collections and higher-order functions
Enums, tuples, records, all immutable
Constraints on types and functions

Stateful Features

Boxes (like Clojure’s ref)
Transactional Memory
State Machines
Interactors

Extensible/Embeddable through modular language implementation and other means.
(Meta-) Tooling

Language Workbench

Open Source, by JetBrains

Very Powerful

Used for years by itemis and others

Vast Experience
MPS: Language Toolkit

- Refactorings, Find Usages, Syntax Coloring, Debugging, ...
MPS: Notational Freedom
Verifying Infrastructure
An Architecture For Smart Contracts

Executable Multi-Party Contract Language

MPS / Convecton

KernelF

Iulia
Ethereum VM/Network

Java
Hyperledger

HTML
JavaScript
CSS

SMTLIB
Z3

Distribution, Trust
Interaction, Integration
Correctness
Verifying Blockchain Infrastructure

Formal Semantics of the EVM in K

IELE: Register-Based VM for the Blockchain

ERC20-K: Formal Executable Spec of ERC20
https://github.com/runtimeverification/erc20-semantics

Formal Verification for Solidity Contracts
https://forum.ethereum.org/discussion/3779/formal-verification-for-solidity-contracts
An Architecture For Smart Contracts

Did I program/specify the right behaviors?

Will the infrastructure execute the behaviors faithfully?
An Architecture For Smart Contracts

Validation

Ensure that the program does the correct things, wrt. to the requirements.

Did I program/specify the right behaviors?

Verification

Ensure that the program performs correctly the things the program text tells it to do.

Will the infrastructure execute the behaviors faithfully?
Wrap Up
Further Reading

**Mutable State in KernelF**
https://medium.com/@markusvoelter/dealing-with-mutable-state-in-kernelf-e0fdec8a489b

**A Smart Contract Development Stack**
https://languageengineering.io/a-smart-contract-development-stack-54533a3a503a

**A Smart Contract Development Stack, Pt. 2**
https://languageengineering.io/a-smart-contract-development-stack-part-ii-game-theoretical-aspects-ca7a9d2e548d

**KernelF Reference**
http://voelter.de/data/pub/kernelf-reference.pdf

**DSLs in Safety-Critical Development**
http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf
Contracts must be functionally correct in order for stakeholders to trust them.

We need better languages to describe contracts in a meaningful way.

Integration of verification tools can be an important step to assure correctness.

Simulation, Experimentation and Test should be available in an interactive, local environment.

Deployment to Blockchain is non-func, it provides guarantees beyond functionality.

Other deployments are useful, that provide other trade-offs (secure↓, fast↑).