Functional Programming and Finance

Philip Wadler
University of Edinburgh / IOHK

IO

Haskell and Finance

seL4

Security. Performance. Proof.

ABN·AMRO

Bank of America

CREDIT SUISSE

BARCLAYS

TSURU CAPITAL

Standard Chartered

Deutsche Bank

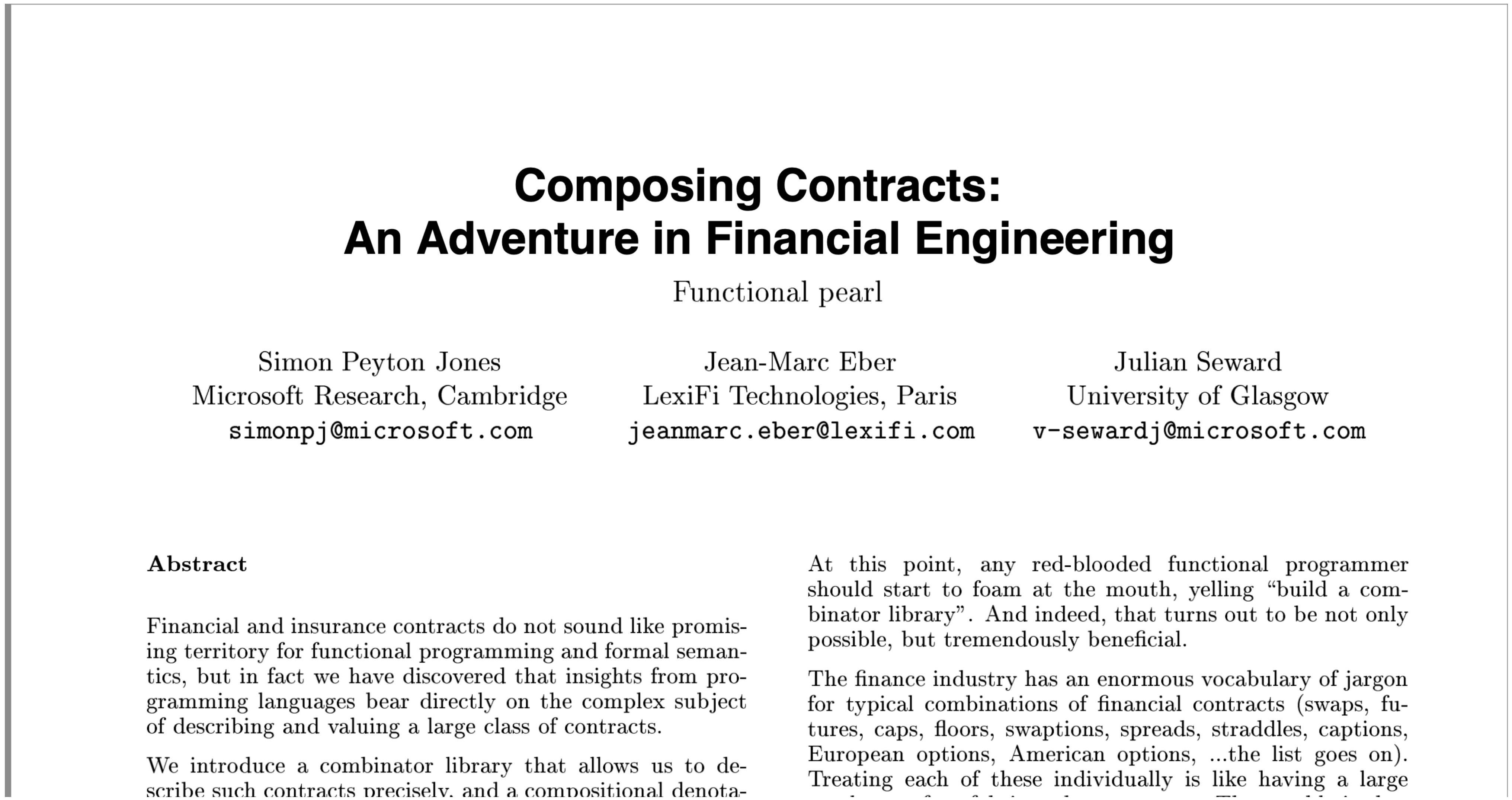
TSURU CAPITAL

Haskell / Rust

O’Caml and Finance

Jane Street

Domain-Specific Languages



Composing Contracts: An Adventure in Financial Engineering

Functional pearl

Simon Peyton Jones
Microsoft Research, Cambridge
simonpj@microsoft.com

Jean-Marc Eber
LexiFi Technologies, Paris
jeanmarc.eber@lexifi.com

Julian Seward
University of Glasgow
v-sewardj@microsoft.com

Abstract

Financial and insurance contracts do not sound like promising territory for functional programming and formal semantics, but in fact we have discovered that insights from programming languages bear directly on the complex subject of describing and valuing a large class of contracts.

We introduce a combinator library that allows us to describe such contracts precisely, and a compositional denota-

At this point, any red-blooded functional programmer should start to foam at the mouth, yelling "build a combinator library". And indeed, that turns out to be not only possible, but tremendously beneficial.

The finance industry has an enormous vocabulary of jargon for typical combinations of financial contracts (swaps, futures, caps, floors, swaptions, spreads, straddles, captions, European options, American options, ...the list goes on). Treating each of these individually is like having a large

International Conference on Functional Programming, Sep 2000

LexiFi

Digital Asset

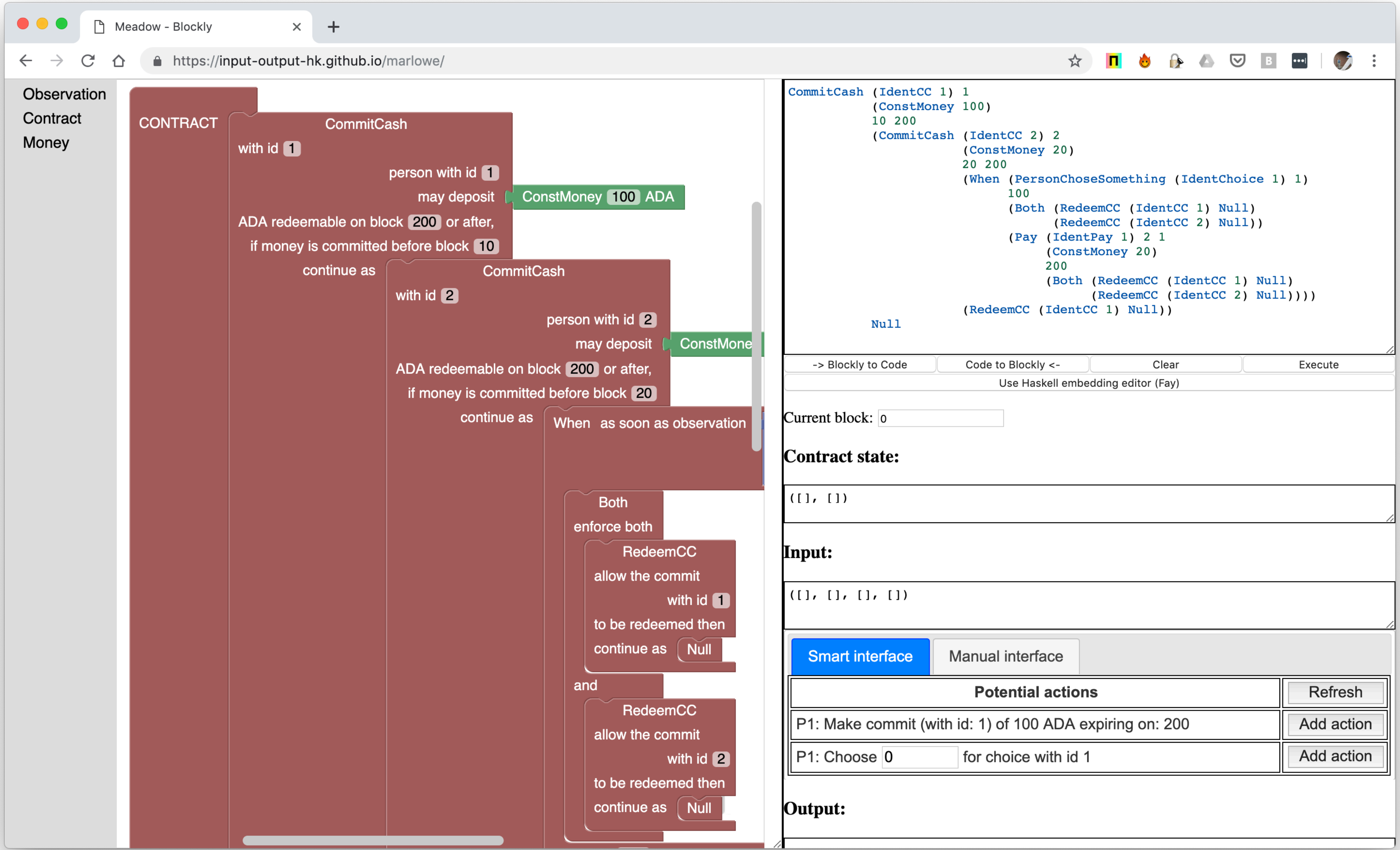
DAML

DEON DIGITAL

Business Engineer

I|O

Marlowe



CONTRACT
CommitCash
with id 1
person with id 1
may deposit ConstMoney 100 ADA
ADA redeemable on block 200 or after,
if money is committed before block 10
continue as
CommitCash
with id 2
person with id 2
may deposit ConstMone
ADA redeemable on block 200 or after,
if money is committed before block 20
continue as
When as soon as observation
Both
enforce both
RedeemCC
allow the commit
with id 1
to be redeemed then
continue as Null
and
RedeemCC
allow the commit
with id 2
to be redeemed then
continue as Null

```
CommitCash (IdentCC 1) 1
           (ConstMoney 100)
           10 200
           (CommitCash (IdentCC 2) 2
                       (ConstMoney 20)
                       20 200
                       (When (PersonChoseSomething (IdentChoice 1) 1)
                             100
                             (Both (RedeemCC (IdentCC 1) Null)
                                   (RedeemCC (IdentCC 2) Null))
                             (Pay (IdentPay 1) 2 1
                                  (ConstMoney 20)
                                  200
                                  (Both (RedeemCC (IdentCC 1) Null)
                                        (RedeemCC (IdentCC 2) Null))))
                       (RedeemCC (IdentCC 1) Null))
           Null
```

Current block: 0

Contract state:

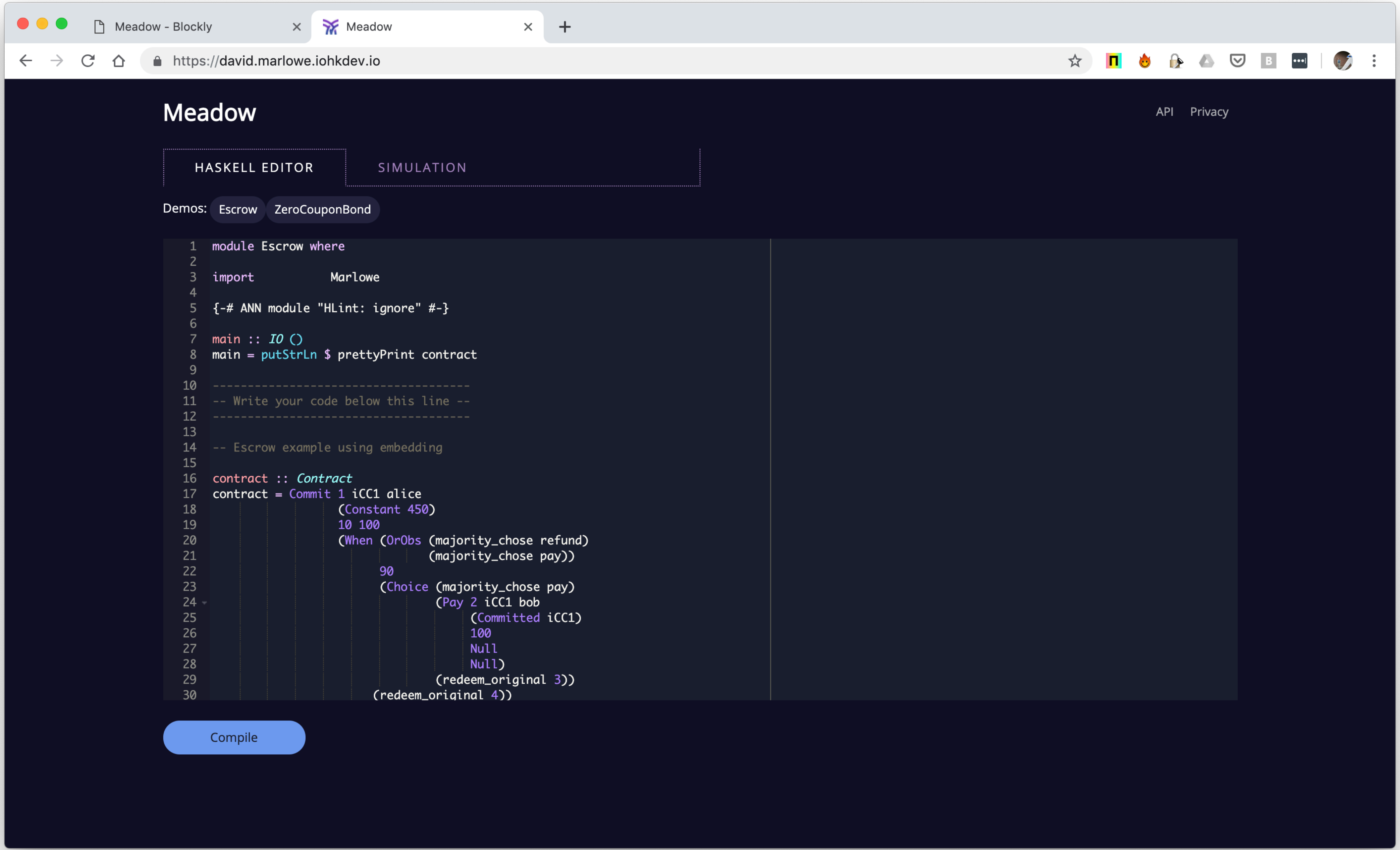
```
([], [])
```

Input:

```
([], [], [], [])
```

Potential actions	Refresh
P1: Make commit (with id: 1) of 100 ADA expiring on: 200	Add action
P1: Choose 0 for choice with id 1	Add action

Output:

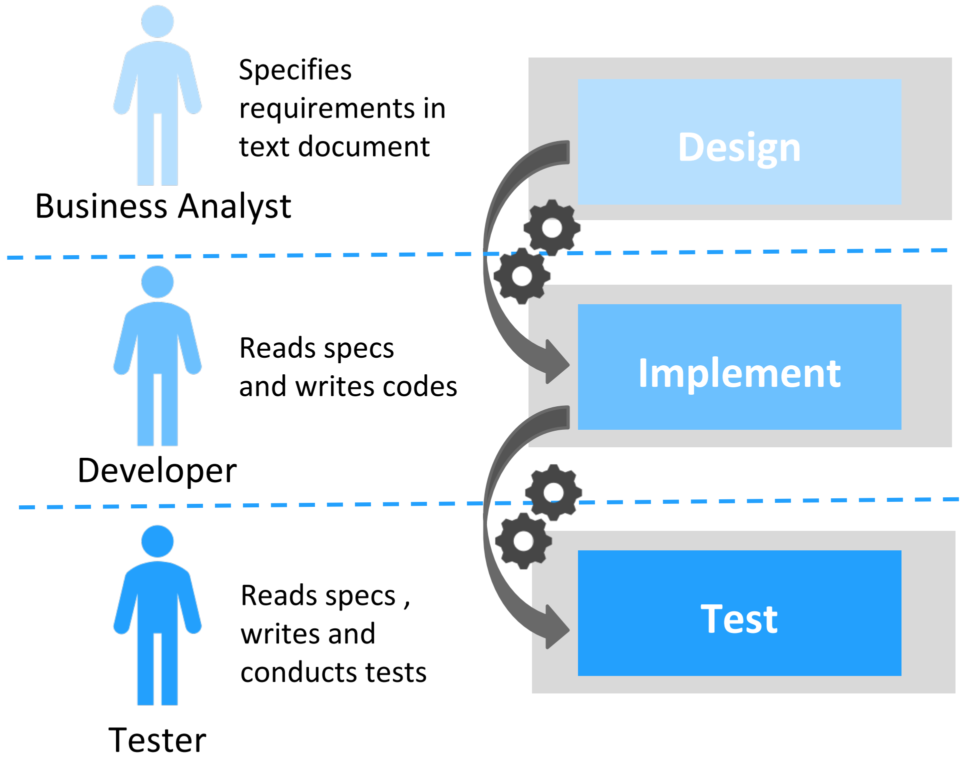


```haskell
module Escrow where

import           Marlowe

{-# ANN module "HLint: ignore" #-}

main :: IO ()
main = putStrLn $ prettyPrint contract

-------------------------------------
-- Write your code below this line --
-------------------------------------

-- Escrow example using embedding

contract :: Contract
contract = Commit 1 iCC1 alice
                  (Constant 450)
                  10 100
                  (When (OrObs (majority_chose refund)
                               (majority_chose pay))
                        90
                        (Choice (majority_chose pay)
                                (Pay 2 iCC1 bob
                                     (Committed iCC1)
                                     100
                                     Null
                                     Null)
                                (redeem_original 3))
                        (redeem_original 4))
```

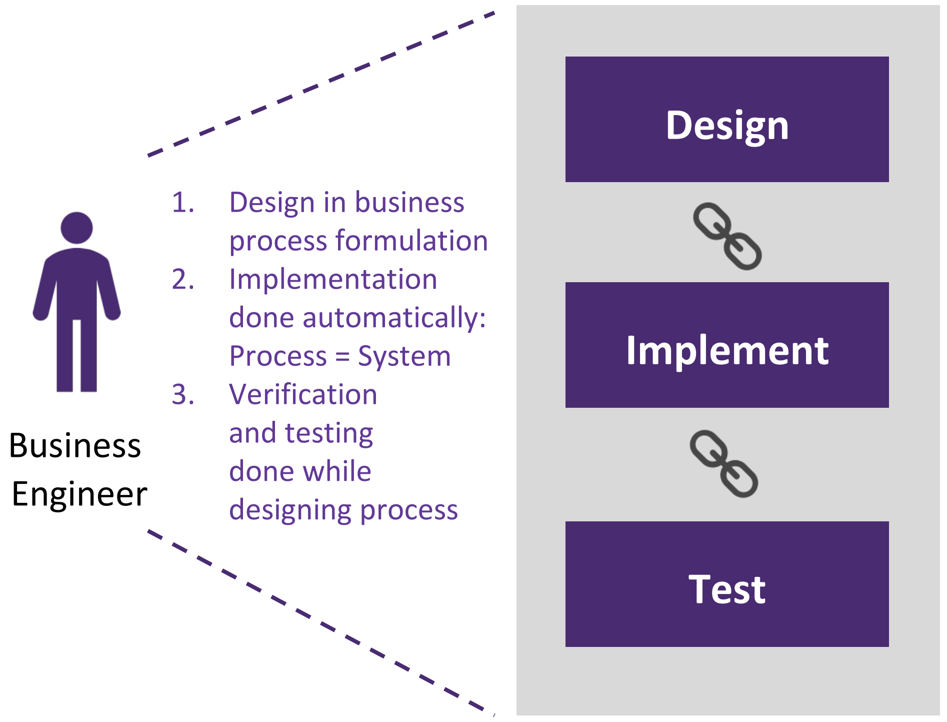
ADVANTAGE
SINGLE
SOURCE
OF TRUTH

Eliminate gap between design, implementation and testing

Traditional approach


Specifies requirements in text document
Business Analyst
Design
Reads specs and writes codes
Developer
Implement
Reads specs , writes and conducts tests
Tester
Test

Deon Digital approach



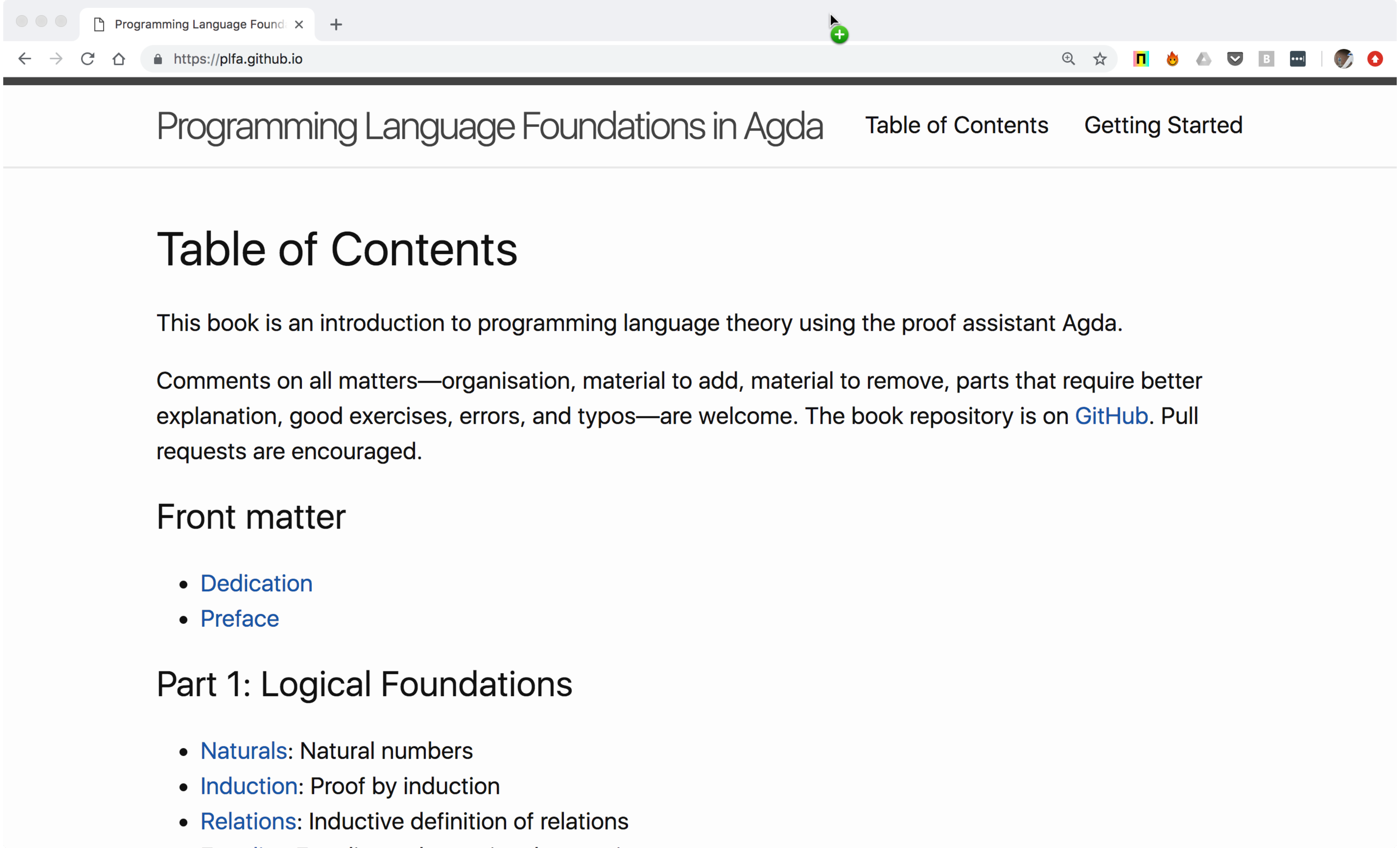
1. Design in business process formulation
2. Implementation done automatically: Process = System
3. Verification and testing done while designing process

DEON DIGITAL

I|O

Formal Methods

Coq



Programming Language Foundations in Agda Table of Contents Getting Started

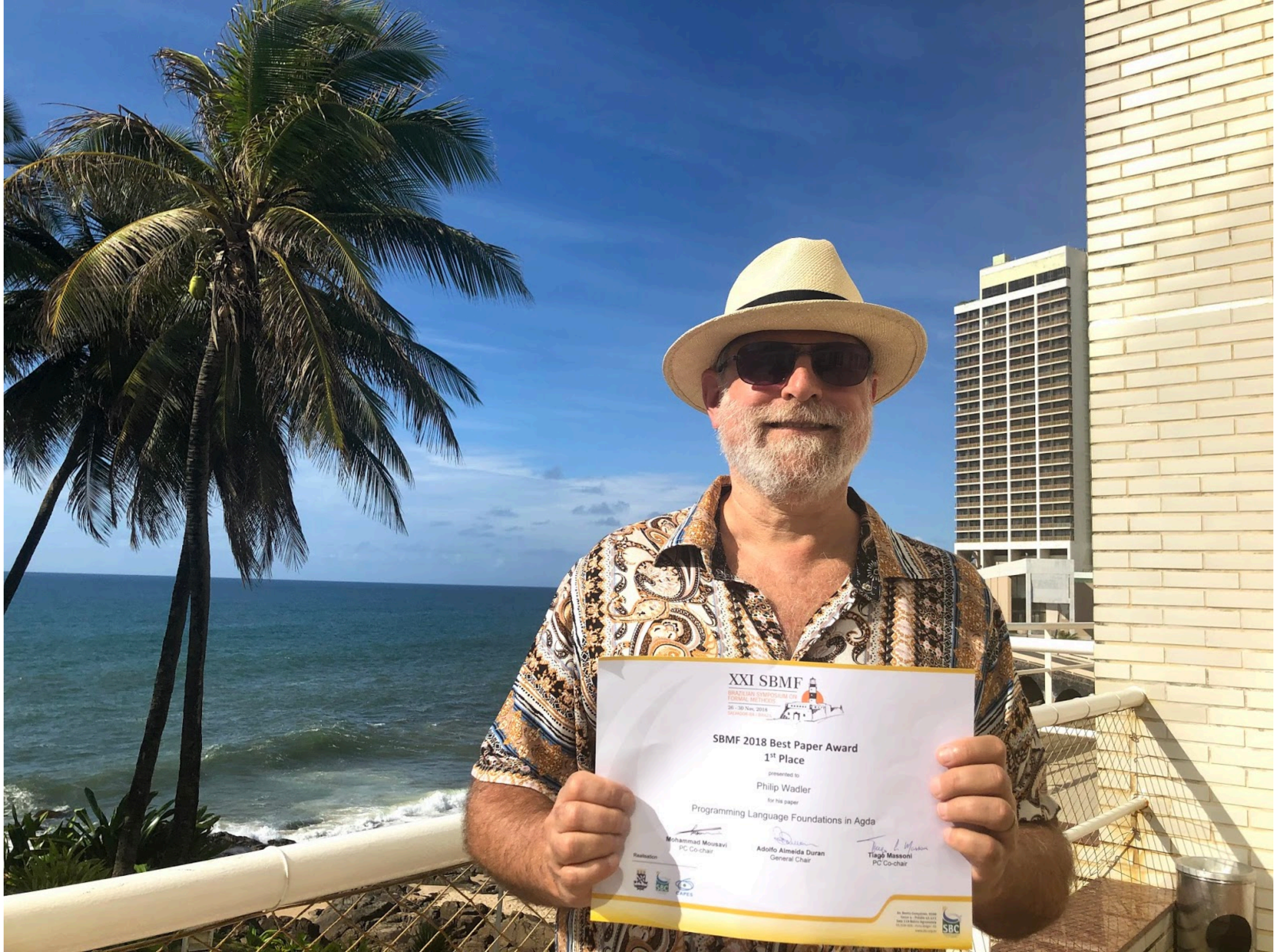
Table of Contents

This book is an introduction to programming language theory using the proof assistant Agda.

Comments on all matters—organisation, material to add, material to remove, parts that require better explanation, good exercises, errors, and typos—are welcome. The book repository is on GitHub. Pull requests are encouraged.

Front matter

Part 1: Logical Foundations


XXI SBMF

BRAZILIAN SYMPOSIUM ON FORMAL METHODS

26 - 30 Nov, 2018

SALVADOR-BA / BRAZIL

SBMF 2018 Best Paper Award

1st Place

presented to

Philip Wadler

for his paper

Programming Language Foundations in Agda

Mohammad Mousavi
PC Co-chair

Adolfo Almeida Duran
General Chair

Tiago Massoni
PC Co-chair

Realisation

IMANDRA

REASONING AS A SERVICE®