MAKER

presents the

REFERENCE IMPLEMENTATION

of the remarkable

DAI CREDIT SYSTEM

issuing a diversely collateralized stablecoin

with last update on March 9, 2017.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The DAI CREDIT SYSTEM, henceforth also "Maker," is a network of Ethereum contracts designed to issue the DAI currency token and automatically adjust incentives in order to keep dai market value stable relative to SDR[1] in the short and medium term.

New dai enters the money supply when a borrower takes out a loan backed by an excess of collateral locked in Maker's token vault. The debt and collateral amounts are recorded in a *collateralized debt position*, or CDP. Thus all outstanding dai represents some CDP owner's claim on their collateral.

Maker's knowledge of the market values of dai and the various tokens used as collateral comes from *price feeds*. Prices are used to continuously assess the risk of each CDP. If the value of a CDP's collateral drops below a certain multiple of its debt, it is marked for liquidation, which triggers a decentralized auction mechanism.

Another token, MKR, is also controlled by Maker, acting as a "share" in the system itself. When a CDP liquidation fails to recover the full value of debt, Maker mints more MKR and auctions it out. Thus MKR is used to fund last resort market making. The value of the MKR token is based on the *stability fee* imposed on all dai loans: stability fee revenue goes toward buying MKR for burning.

This document is an executable technical specification of the exact workings of the Maker smart contracts.

[1]"Special Drawing Rights" (ticker symbol XDR), the international reserve asset created by the International Monetary Fund, whose value is derives from a weighted basket of world currencies. In the long term, the value of dai may diverge from the value of SDR; whether in an inflationary or deflationary way will depend on market forces.

1.1 Reference implementation

The version of this system that will be deployed on the Ethereum blockchain is written in Solidity, which is a workable smart contract implementation language. This reference implementation is a precise model of the behavior of those contracts, written as a "literate" Haskell program. The motivations for such a reference implementation include:

1. **Comparison.** Checking two free-standing implementations against each other is a well-known way of ensuring that they both behave as intended.

2. **Testing.** Haskell lets us use flexible and powerful testing tools such as QuickCheck and SmallCheck for comprehensively verifying key properties as a middle ground between unit testing and formal verification.

3. **Explicitness.** Coding the contract behavior in Haskell, a purely functional language, enforces explicit description of aspects which Solidity leaves implicit. For example, a Solidity program can read previously unwritten storage and get back a zero value, whereas in Haskell we must give explicit defaults. The state rollback behavior of failed actions is also explicit in the type of the execution function, which may return an error.

4. **Type correctness.** While Solidity does have a static type system, it is not expressive enough to encode the distinctions made by our system. In particular, the two different decimal fixed point number types that we use are typed in Solidity with one and the same `uint128` type. In Haskell we can make this distinction explicit.

5. **Formality.** The work of translating a Solidity program into a purely functional program opens up opportunities for certain types of formal verification. In particular, this document will be useful for modelling aspects of the system in a proof assistant like Agda, Idris, Coq, or Isabelle. We can also use logical tools for Haskell, such as Liquid Haskell (which provides compile time logical property checking) and `sbv` (a toolkit for model checking and symbolic execution).

6. **Simulation.** Solidity is highly specific to the Ethereum blockchain environment and as such does not have facilities for interfacing with files or other computer programs. This makes the Solidity implementation of the system less useful for doing simulations of the system's economic, game-theoretic, or statistical aspects.

Part I

Implementation

Chapter 2

Preamble

We replace the default prelude module with our own. This brings in dependencies and hides unneeded symbols. Consult Appendix A to see exactly what is brought into scope.

```haskell
module Maker where

import Maker.Prelude   -- Fully import the Maker prelude
import Prelude ()      -- Import nothing from Prelude

import Maker.Decimal

import Debug.Trace
```

Chapter 3

Types

3.1 Numeric types

The system uses two different precisions of decimal fixed point numbers, which we call *wads* and *rays*, having respectively 18 digits of precision (used for token quantities) and 36 digits (used for precise rates and ratios).

Define the distinct `wad` type for currency quantities

newtype `Wad` = `Wad` (Decimal E18)
  deriving (Ord, Eq, Num, Real, Fractional, RealFrac)

Define the distinct `ray` type for precise rate quantities

newtype `Ray` = `Ray` (Decimal E36)
  deriving (Ord, Eq, Num, Real, Fractional, RealFrac)

See Appendix B for details on how we modify Haskell's decimal fixed point type to do more correct rounding for multiplication and division.

Haskell number types are not automatically converted, so in calculations that combine wads and rays, we convert explicitly with a *cast* function.

Convert via fractional n/m form.

$cast :: (\text{Real } a, \text{Fractional } b) \Rightarrow a \rightarrow b$
$cast = \mathit{fromRational} \circ \mathit{toRational}$

We also define a type for time durations in whole seconds.

newtype `Sec` = `Sec` Int
  deriving (Eq, Ord, Enum, Num, Real, Integral)

instance Epsilon `Wad` **where** ϵ = `Wad` ϵ
instance Epsilon `Ray` **where** ϵ = `Ray` ϵ

3.2 Identifiers and addresses

There are several kinds of identifiers used in the system, and we can use types to distinguish them.

The type parameter a creates distinct types.
For example, Id Foo and Id Bar are incompatible.

data Id a = Id String
  deriving (Show, Eq, Ord)

We define another type for representing Ethereum account addresses.

data Address = Address String
  deriving (Ord, Eq, Show)

We also have three predefined entity identifiers.

The DAI token address
id_{DAI} = Id `"DAI"`

The CDP engine address
$id_{\texttt{vat}}$ = Address `"VAT"`

The account with ultimate authority
◊ *Kludge until authority is modelled*
id_{god} = Address `"GOD"`

The address of the test driver
id_{toy} = Address `"TOY"`

This section introduces the records stored by the Maker system.

3.3 `Gem` — ERC20 token model

data Holder = AddressHolder Address
  | JarHolder (Id `Jar`)
  deriving (Eq, Show, Ord)

```haskell
data Gem = Gem {
  gemTotalSupply :: Wad,
  gemBalanceOf   :: Map Holder Wad
  } deriving (Eq, Show)
```

3.4 Jar — collateral type

```haskell
data Jar = Jar {
  jarGem :: Gem,  -- Collateral token
  jarTag :: Wad,  -- Market price
  jarZzz :: Sec   -- Price expiration
  } deriving (Eq, Show)
```

3.5 Ilk — CDP type

```haskell
data Ilk = Ilk {
  ilkJar :: Id Jar,  -- Collateral vault
  ilkAxe :: Ray,     -- Liquidation penalty
  ilkHat :: Wad,     -- Debt ceiling
  ilkMat :: Ray,     -- Liquidation ratio
  ilkTax :: Ray,     -- Stability fee
  ilkLag :: Sec,     -- Limbo duration
  ilkRho :: Sec,     -- Last dripped
  ilkRum :: Wad,     -- Total debt in debt unit
  ilkChi :: Ray      -- Dai value of debt unit
  } deriving (Eq, Show)
```

3.6 Urn — collateralized debt position (CDP)

```haskell
data Urn = Urn {
  urnCat :: Maybe Address,  -- Address of liquidation initiator
```

$urnVow$ :: Maybe Address, Address of liquidation contract
$urnLad$ :: Address, Issuer
$urnIlk$ :: Id `Ilk`, CDP type
$urnArt$ :: `Wad`, Outstanding debt in debt unit
$urnJam$ :: `Wad` Collateral amount in debt unit
} **deriving** (Eq, Show)

3.7 `Vat` — CDP engine

data `Vat` = `Vat` {
$vatFix$ :: `Wad`, Market price
$vatHow$ :: `Ray`, Sensitivity
$vatPar$ :: `Wad`, Target price
$vatWay$ :: `Ray`, Target rate
$vatTau$ :: `Sec`, Last prodded
$vatJoy$ :: `Wad`, Unprocessed stability fees
$vatSin$ :: `Wad`, Bad debt from liquidated CDPs
$vatJars$ :: Map (Id `Jar`) `Jar`, Collateral tokens
$vatIlks$ :: Map (Id `Ilk`) `Ilk`, CDP types
$vatUrns$ :: Map (Id `Urn`) `Urn` CDPs
} **deriving** (Eq, Show)

3.8 System model

data System = System {
$systemVat$ :: `Vat`, Root Maker entity
$systemEra$ :: `Sec`, Current time stamp
$systemSender$ :: Address, Sender of current act
$systemAccounts$ :: [Address] For test suites
} **deriving** (Eq, Show)

Lens fields

```
makeFields ''Gem
makeFields ''Jar
makeFields ''Ilk
makeFields ''Urn
makeFields ''Vat
makeFields ''System
```

3.9 Default data

```
defaultIlk :: Id Jar → Ilk
defaultIlk id_jar = Ilk {
  ilkJar  = id_jar,
  ilkAxe  = Ray 1,
  ilkMat  = Ray 1,
  ilkTax  = Ray 1,
  ilkHat  = Wad 0,
  ilkLag  = Sec 0,
  ilkChi  = Ray 1,
  ilkRum  = Wad 0,
  ilkRho  = Sec 0
}
```

```
defaultUrn :: Id Ilk → Address → Urn
defaultUrn id_ilk id_lad = Urn {
  urnVow = Nothing,
  urnCat = Nothing,
  urnLad = id_lad,
  urnIlk = id_ilk,
  urnArt = Wad 0,
  urnJam = Wad 0
}
```

```
initialVat :: Ray → Vat
initialVat how_0 = Vat {
```

$$
\begin{aligned}
&\quad \mathit{vatTau} &&= 0,\\
&\quad \mathit{vatFix} &&= \texttt{Wad}\ 1,\\
&\quad \mathit{vatPar} &&= \texttt{Wad}\ 1,\\
&\quad \mathit{vatHow} &&= \texttt{how}_0,\\
&\quad \mathit{vatWay} &&= \texttt{Ray}\ 1,\\
&\quad \mathit{vatJoy} &&= \texttt{Wad}\ 0,\\
&\quad \mathit{vatSin} &&= \texttt{Wad}\ 0,\\
&\quad \mathit{vatIlks} &&= \varnothing,\\
&\quad \mathit{vatUrns} &&= \varnothing,\\
&\quad \mathit{vatJars} &&=\\
&\qquad \mathit{singleton}\ \mathit{id}_{\text{DAI}}\ \texttt{Jar}\ \{\\
&\qquad\quad \mathit{jarGem} = \texttt{Gem}\ \{\\
&\qquad\qquad \mathit{gemTotalSupply} = 0,\\
&\qquad\qquad \mathit{gemBalanceOf} \;\;= \varnothing\\
&\qquad\quad \},\\
&\qquad\quad \mathit{jarTag} = \texttt{Wad}\ 0,\\
&\qquad\quad \mathit{jarZzz} \;= 0\\
&\qquad \}\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{initialSystem} :: \texttt{Ray} \to \text{System}\\
&\mathit{initialSystem}\ \texttt{how}_0 = \text{System}\ \{\\
&\quad \mathit{systemVat} &&= \mathit{initialVat}\ \texttt{how}_0,\\
&\quad \mathit{systemEra} &&= 0,\\
&\quad \mathit{systemSender} &&= \mathit{id}_{god},\\
&\quad \mathit{systemAccounts} &&= \mathit{mempty}\\
&\}
\end{aligned}
$$

Chapter 4

Acts

The *acts* are the basic state transitions of the credit system.

For details on the underlying “Maker monad,” which specifies how the act definitions behave with regard to state and rollback thereof, see chapter 5.

4.1 Assessment

We divide an urn's situation into five stages of risk. Table 4.1 shows which acts each stage allows. The stages are naturally ordered from more to less risky.

$$\textbf{data}\ \text{Stage} = \texttt{Dread} \mid \texttt{Grief} \mid \texttt{Panic} \mid \texttt{Worry} \mid \texttt{Anger} \mid \texttt{Pride}$$
$$\quad \textbf{deriving}\ (\text{Eq}, \text{Ord}, \text{Show})$$

First we define a pure function *analyze* that determines an urn's stage.

$$
\begin{array}{l}
\textit{analyze}\ \texttt{era}_0\ \texttt{par}_0\ \texttt{urn}_0\ \texttt{ilk}_0\ \texttt{jar}_0 = \\
\quad \textbf{if} \\
\quad\ \ \text{Undergoing liquidation?} \\
\quad\quad \mid \textit{view}\ \texttt{vow}\ \texttt{urn}_0 \not\equiv \text{Nothing} \hfill \to \texttt{Dread} \\
\quad\ \ \text{Liquidation triggered?} \\
\quad\quad \mid \textit{view}\ \texttt{cat}\ \texttt{urn}_0 \not\equiv \text{Nothing} \hfill \to \texttt{Grief} \\
\quad\ \ \text{Undercollateralized?} \\
\quad\quad \mid \texttt{pro} < \texttt{min} \hfill \to \texttt{Panic} \\
\quad\ \ \text{Price feed expired?} \\
\quad\quad \mid \texttt{era}_0 > \textit{view}\ \texttt{zzz}\ \texttt{jar}_0 + \textit{view}\ \texttt{lag}\ \texttt{ilk}_0 \to \texttt{Panic} \\
\quad\ \ \text{Price feed in limbo?} \\
\quad\quad \mid \textit{view}\ \texttt{zzz}\ \texttt{jar}_0 < \texttt{era}_0 \hfill \to \texttt{Worry} \\
\quad\ \ \text{Debt ceiling reached?} \\
\quad\quad \mid \texttt{cap} > \textit{view}\ \texttt{hat}\ \texttt{ilk}_0 \hfill \to \texttt{Anger} \\
\quad\ \ \text{Safely overcollateralized} \\
\quad\quad \mid \textit{otherwise} \hfill \to \texttt{Pride} \\
\quad \textbf{where} \\
\quad\ \ \text{CDP's collateral value in SDR:} \\
\quad\ \ \texttt{pro} = \textit{view}\ \texttt{jam}\ \texttt{urn}_0 * \textit{view}\ \texttt{tag}\ \texttt{jar}_0 \\
\quad\ \ \text{CDP type's total debt in SDR:} \\
\quad\ \ \texttt{cap} = (\textit{view}\ \texttt{rum}\ \texttt{ilk}_0 * \textit{cast}\ (\textit{view}\ \texttt{chi}\ \texttt{ilk}_0)) :: \texttt{Wad} \\
\quad\ \ \text{CDP's debt in SDR:} \\
\quad\ \ \texttt{con} = \textit{view}\ \texttt{art}\ \texttt{urn}_0 * \textit{cast}\ (\textit{view}\ \texttt{chi}\ \texttt{ilk}_0) * \texttt{par}_0 \\
\quad\ \ \text{Required collateral as per liquidation ratio:} \\
\quad\ \ \texttt{min} = \texttt{con} * \textit{view}\ \texttt{mat}\ \texttt{ilk}_0
\end{array}
$$

Table 4.1: Urn acts in the five stages of risk

	give	shut	lock	wipe	free	draw	bite	grab	plop	
Pride	•	•	•	•	•	•				overcollateralized
Anger	•	•	•	•	•					debt ceiling reached
Worry	•	•	•	•						price feed in limbo
Panic	•	•	•	•			•			undercollateralized
Grief	•							•		liquidation initiated
Dread	•								•	liquidation in progress

Now we define the internal act `gaze` which returns the value of *analyze* after ensuring the system state is updated.

$\texttt{gaze}\ id_{\texttt{urn}} = \textbf{do}$

Perform dai revaluation and rate adjustment

$\texttt{prod}$

Update price of specific debt unit

$id_{\texttt{ilk}} \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{ilk})$

$\texttt{drip}\ id_{\texttt{ilk}}$

Read parameters for risk analysis

$\texttt{era}_0 \leftarrow \mathit{use}\ \texttt{era}$

$\texttt{par}_0 \leftarrow \mathit{use}\ (\texttt{vat} \circ \texttt{par})$

$\texttt{urn}_0 \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}})$

$\texttt{ilk}_0 \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{ilk}s \circ \mathit{ix}\ (\mathit{view}\ \texttt{ilk}\ \texttt{urn}_0))$

$\texttt{jar}_0 \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{jar}s \circ \mathit{ix}\ (\mathit{view}\ \texttt{jar}\ \texttt{ilk}_0))$

Return risk stage of CDP

$\mathit{return}\ (\mathit{analyze}\ \texttt{era}_0\ \texttt{par}_0\ \texttt{urn}_0\ \texttt{ilk}_0\ \texttt{jar}_0)$

4.2 Lending

$\texttt{open}\ id_{\texttt{urn}}\ id_{\texttt{ilk}} =$

$\textbf{do}$

$id_{\texttt{lad}} \leftarrow \mathit{use\ sender}$

$\texttt{vat} \circ \texttt{urn}s \circ \mathit{at}\ id_{\texttt{urn}} \mathrel{?=} \mathit{defaultUrn}\ id_{\texttt{ilk}}\ id_{\texttt{lad}}$

$\texttt{lock}\ id_{\texttt{urn}}\ x = \textbf{do}$

Ensure CDP exists; identify collateral type

$id_{\texttt{ilk}} \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{ilk})$
$id_{\texttt{jar}} \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{jar})$

Record an increase in collateral

$\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{jam} \mathrel{+}= x$

Take sender's tokens

$id_{\texttt{lad}} \leftarrow use\ sender$
$\texttt{pull}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ x$

$\texttt{free}\ id_{\texttt{urn}}\ \texttt{wad}_{\texttt{gem}} = \mathbf{do}$

Fail if sender is not the CDP owner

$id_{sender} \leftarrow use\ sender$
$id_{\texttt{lad}} \quad \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{lad})$
$\texttt{aver}\ (id_{sender} \equiv id_{\texttt{lad}})$

Decrease the collateral amount

$\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{jam} \mathrel{-}= \texttt{wad}_{\texttt{gem}}$

Roll back if undercollateralized

$\texttt{gaze}\ id_{\texttt{urn}} \ggg= \texttt{aver} \circ (\equiv \texttt{Pride})$

Send the collateral to the CDP owner

$id_{\texttt{ilk}} \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{ilk})$
$id_{\texttt{jar}} \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{jar})$
$\texttt{push}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ \texttt{wad}_{\texttt{gem}}$

$\texttt{draw}\ id_{\texttt{urn}}\ \texttt{wad}_{\text{DAI}} = \mathbf{do}$

Fail if sender is not the CDP owner

$id_{sender} \leftarrow use\ sender$
$id_{\texttt{lad}} \quad \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{lad})$
$\texttt{aver}\ (id_{sender} \equiv id_{\texttt{lad}})$

Update value of debt unit

$id_{\texttt{ilk}} \quad \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{ilk})$
$\texttt{chi}_1 \quad \leftarrow \texttt{drip}\ id_{\texttt{ilk}}$

Denominate draw amount in debt unit

$\mathbf{let}\ \texttt{wad}_{\texttt{chi}} = \texttt{wad}_{\text{DAI}}\ /\ cast\ \texttt{chi}_1$

Increase debt

$\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{art} \mathrel{+}= \texttt{wad}_{\texttt{chi}}$

Roll back unless overcollateralized

$\texttt{gaze}\ id_{\texttt{urn}} \ggg \texttt{aver} \circ (\equiv \texttt{Pride})$

Mint dai and send to the CDP owner

$\texttt{mint}\ id_{\text{DAI}}\ \texttt{wad}_{\text{DAI}}$
$\texttt{push}\ id_{\text{DAI}}\ id_{\texttt{lad}}\ \texttt{wad}_{\text{DAI}}$

$\texttt{wipe}\ id_{\texttt{urn}}\ \texttt{wad}_{\text{DAI}} = \textbf{do}$

Fail if sender is not the CDP owner

$id_{sender} \leftarrow use\ sender$
$id_{\texttt{lad}} \quad \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{lad})$
$\texttt{aver}\ (id_{sender} \equiv id_{\texttt{lad}})$

Update value of debt unit

$id_{\texttt{ilk}} \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{ilk})$
$\texttt{chi}_1 \leftarrow \texttt{drip}\ id_{\texttt{ilk}}$

Roll back unless overcollateralized

$\texttt{gaze}\ id_{\texttt{urn}} \ggg \texttt{aver} \circ (\equiv \texttt{Pride})$

Denominate dai amount in debt unit

$\textbf{let}\ \texttt{wad}_{\texttt{chi}} = \texttt{wad}_{\text{DAI}}\ /\ cast\ \texttt{chi}_1$

Reduce debt

$\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{art} \mathrel{-=} \texttt{wad}_{\texttt{chi}}$

Take dai from CDP owner, or roll back

$\texttt{pull}\ id_{\text{DAI}}\ id_{\texttt{lad}}\ \texttt{wad}_{\text{DAI}}$

Destroy dai

$\texttt{burn}\ id_{\text{DAI}}\ \texttt{wad}_{\text{DAI}}$

$\texttt{give}\ id_{\texttt{urn}}\ id_{\texttt{lad}} = \textbf{do}$

$x \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{lad})$
$y \leftarrow use\ sender$
$\texttt{aver}\ (x \equiv y)$
$\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{lad} := id_{\texttt{lad}}$

$\texttt{shut}\ id_{\texttt{urn}} = \textbf{do}$

Update value of debt unit

$id_{\texttt{ilk}} \leftarrow look\ (\texttt{vat} \circ \texttt{urn}s \circ ix\ id_{\texttt{urn}} \circ \texttt{ilk})$
$\texttt{chi}_1 \leftarrow \texttt{drip}\ id_{\texttt{ilk}}$

Attempt to repay all the CDP's outstanding dai

$\texttt{art}_0 \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ \mathit{id}_{\texttt{urn}} \circ \texttt{art})$

$\texttt{wipe}\ \mathit{id}_{\texttt{urn}}\ (\texttt{art}_0 * \mathit{cast}\ \texttt{chi}_1)$

Reclaim all the collateral

$\mathit{jam0} \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ \mathit{id}_{\texttt{urn}} \circ \texttt{jam})$

$\texttt{free}\ \mathit{id}_{\texttt{urn}}\ \mathit{jam0}$

Nullify the CDP

$\texttt{vat} \circ \texttt{urn}s \circ \mathit{at}\ \mathit{id}_{\texttt{urn}} := \text{Nothing}$

4.3 Adjustment

$$
\begin{array}{l}
\texttt{prod} = \textbf{do} \\
\quad \texttt{era}_0 \leftarrow \textit{use}\ \texttt{era} \\
\quad \texttt{tau}_0 \leftarrow \textit{use}\ (\texttt{vat} \circ \texttt{tau}) \\
\quad \texttt{fix}_0 \leftarrow \textit{use}\ (\texttt{vat} \circ \texttt{fix}) \\
\quad \texttt{par}_0 \leftarrow \textit{use}\ (\texttt{vat} \circ \texttt{par}) \\
\quad \texttt{how}_0 \leftarrow \textit{use}\ (\texttt{vat} \circ \texttt{how}) \\
\quad \texttt{way}_0 \leftarrow \textit{use}\ (\texttt{vat} \circ \texttt{way}) \\
\quad \textbf{let}
\end{array}
$$

Time difference in seconds

$$age = \texttt{era}_0 - \texttt{tau}_0$$

Current target rate applied to target price

$$\texttt{par}_1 = \texttt{par}_0 * \textit{cast}\ (\texttt{way}_0 \uparrow\uparrow \textit{age})$$

Sensitivity parameter applied over time

$$wag = \texttt{how}_0 * \textit{fromIntegral}\ \textit{age}$$

Target rate scaled up or down

$$
\begin{array}{l}
\texttt{way}_1 = \textit{inj}\ (\textit{prj}\ \texttt{way}_0 + \\
\qquad\qquad \textbf{if}\ \texttt{fix}_0 < \texttt{par}_0\ \textbf{then}\ \textit{wag}\ \textbf{else} - \textit{wag})
\end{array}
$$

$$
\begin{array}{l}
\texttt{vat} \circ \texttt{par} := \texttt{par}_1 \\
\texttt{vat} \circ \texttt{way} := \texttt{way}_1 \\
\texttt{vat} \circ \texttt{tau} := \texttt{era}_0 \\
\textbf{where}
\end{array}
$$

Convert between multiplicative and additive form

$$
\begin{array}{ll}
\textit{prj}\ x & = \textbf{if}\ x \geqslant 1\ \textbf{then}\ x - 1\ \textbf{else}\ 1 - 1 \mathbin{/} x \\
\textit{inj}\ x & = \textbf{if}\ x \geqslant 0\ \textbf{then}\ x + 1\ \textbf{else}\ 1 \mathbin{/} (1 - x)
\end{array}
$$

$\texttt{drip}\ id_{\texttt{ilk}} = \textbf{do}$

Current time stamp

$\texttt{era}_0 \leftarrow use\ \texttt{era}$

Time stamp of previous drip

$\texttt{rho}_0 \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{rho})$

Current stability fee

$\texttt{tax}_0 \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{tax})$

Current value of debt unit

$\texttt{chi}_0 \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{chi})$

Current total debt in debt unit

$rum0 \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{rum})$

Current unprocessed stability fee revenue

$\texttt{joy}_0 \leftarrow look\ (\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{joy})$

$\textbf{let}$

$age = \texttt{era}_0 - \texttt{rho}_0$

$\texttt{chi}_1 = \texttt{chi}_0 * \texttt{tax}_0 \uparrow\uparrow age$

$\texttt{joy}_1 = \texttt{joy}_0 + (cast\ (\texttt{chi}_1 - \texttt{chi}_0) :: \texttt{Wad}) * rum0$

$\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{chi} := \texttt{chi}_1$

$\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{rho} := \texttt{era}_0$

$\texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{joy} := \texttt{joy}_1$

$return\ \texttt{chi}_1$

4.4 Feedback

$\texttt{mark}\ id_{\texttt{jar}}\ \texttt{tag}_1\ \texttt{zzz}_1 =$

$\texttt{auth}\ \$\ \textbf{do}$

$\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{tag} := \texttt{tag}_1$

$\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{zzz} := \texttt{zzz}_1$

$\texttt{tell}\ x =$

$\texttt{auth}\ \$\ \textbf{do}$

$\texttt{vat} \circ \texttt{fix} := x$

4.5 Liquidation

$\texttt{bite}\ id_{\texttt{urn}} = \textbf{do}$

Fail if CDP is not in need of liquidation

$\texttt{gaze}\ id_{\texttt{urn}} \ggg\!= \texttt{aver} \circ (\equiv \texttt{Panic})$

Record the sender as the liquidation initiator

$id_{\texttt{cat}} \leftarrow \mathit{use\ sender}$

$\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{cat} := id_{\texttt{cat}}$

Read current debt

$\texttt{art}_0 \quad \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{art})$

Update value of debt unit

$id_{\texttt{ilk}} \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{ilk})$

$\texttt{chi}_1 \leftarrow \texttt{drip}\ id_{\texttt{ilk}}$

Read liquidation penalty ratio

$id_{\texttt{ilk}} \quad \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{ilk})$

$\texttt{axe}_0 \quad \leftarrow \mathit{look}\ (\texttt{vat} \circ \texttt{ilk}s \circ \mathit{ix}\ id_{\texttt{ilk}} \circ \texttt{axe})$

Apply liquidation penalty to debt

$\textbf{let}\ \texttt{art}_1 = \texttt{art}_0 * \texttt{axe}_0$

Update CDP debt

$\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{art} := \texttt{art}_1$

Record as bad debt

$\texttt{sin} \mathrel{+}= \texttt{art}_1 * \texttt{chi}_1$

$\texttt{grab}\ id_{\texttt{urn}} =$

$\quad \texttt{auth}\ \$\ \textbf{do}$

Fail if CDP is not marked for liquidation

$\texttt{gaze}\ id_{\texttt{urn}} \ggg\!= \texttt{aver} \circ (\equiv \texttt{Grief})$

Record the sender as the CDP's settler

$id_{\texttt{vow}} \leftarrow \mathit{use\ sender}$

$\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{vow} := \mathrm{Just}\ id_{\texttt{vow}}$

Clear the CDP's requester of liquidation

$\texttt{vat} \circ \texttt{urn}s \circ \mathit{ix}\ id_{\texttt{urn}} \circ \texttt{cat} := \mathrm{Nothing}$

$\texttt{heal}\ \texttt{wad}_{\mathrm{DAI}} =$

$\quad \texttt{auth}\ \$\ \textbf{do}$

$\quad\quad \texttt{vat} \circ \texttt{sin} \mathrel{-}= \texttt{wad}_{\mathrm{DAI}}$

$\mathtt{love}\ \mathtt{wad}_{\text{DAI}} =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{joy} \mathrel{-=} \mathtt{wad}_{\text{DAI}}$

4.6 Governance

$\mathtt{form}\ id_{\mathtt{ilk}}\ id_{\mathtt{jar}} =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{ilk}s \circ at\ id_{\mathtt{ilk}} \mathrel{?=} defaultIlk\ id_{\mathtt{jar}}$

$\mathtt{frob}\ how' =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{how} := how'$

$\mathtt{chop}\ id_{\mathtt{ilk}}\ \mathtt{axe}_1 =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{ilk}s \circ ix\ id_{\mathtt{ilk}} \circ \mathtt{axe} := \mathtt{axe}_1$

$\mathtt{cork}\ id_{\mathtt{ilk}}\ \mathtt{hat}_1 =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{ilk}s \circ ix\ id_{\mathtt{ilk}} \circ \mathtt{hat} := \mathtt{hat}_1$

$\mathtt{calm}\ id_{\mathtt{ilk}}\ \mathtt{lag}_1 =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{ilk}s \circ ix\ id_{\mathtt{ilk}} \circ \mathtt{lag} := \mathtt{lag}_1$

$\mathtt{cuff}\ id_{\mathtt{ilk}}\ \mathtt{mat}_1 =$
$\quad \mathtt{auth}\ \$\ \mathbf{do}$
$\qquad \mathtt{vat} \circ \mathtt{ilk}s \circ ix\ id_{\mathtt{ilk}} \circ \mathtt{mat} := \mathtt{mat}_1$

$$
\begin{array}{l}
\texttt{crop}\ id_{\texttt{ilk}}\ \texttt{tax}_1 = \\
\quad \texttt{auth}\ \$\ \mathbf{do} \\
\quad\quad \texttt{drip}\ id_{\texttt{ilk}} \\
\quad\quad \texttt{vat} \circ \texttt{ilk}s \circ ix\ id_{\texttt{ilk}} \circ \texttt{tax} := \texttt{tax}_1
\end{array}
$$

4.7 Treasury

$$
\begin{array}{l}
\texttt{pull}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ w = \mathbf{do} \\
\quad g\ \leftarrow look\ (\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem}) \\
\quad g' \leftarrow transferFrom\ (\text{AddressHolder}\ id_{\texttt{lad}}) \\
\qquad\qquad\qquad\qquad (\text{JarHolder}\ id_{\texttt{jar}})\ w\ g \\
\quad \texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem} := g'
\end{array}
$$

$$
\begin{array}{l}
\texttt{push}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ w = \mathbf{do} \\
\quad g\ \leftarrow look\ (\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem}) \\
\quad g' \leftarrow transferFrom\ (\text{JarHolder}\ id_{\texttt{jar}}) \\
\qquad\qquad\qquad\qquad (\text{AddressHolder}\ id_{\texttt{lad}})\ w\ g \\
\quad \texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem} := g'
\end{array}
$$

$$
\begin{array}{l}
\texttt{mint}\ id_{\texttt{jar}}\ \texttt{wad}_0 = \\
\quad zoom\ (\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem})\ \$\ \mathbf{do} \\
\quad\quad totalSupply \qquad\qquad\qquad\qquad\quad += \texttt{wad}_0 \\
\quad\quad balanceOf \circ ix\ (\text{JarHolder}\ id_{\texttt{jar}}) += \texttt{wad}_0
\end{array}
$$

$$
\begin{array}{l}
\texttt{burn}\ id_{\texttt{jar}}\ \texttt{wad}_0 = \\
\quad zoom\ (\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem})\ \$\ \mathbf{do} \\
\quad\quad totalSupply \qquad\qquad\qquad\qquad\quad -= \texttt{wad}_0 \\
\quad\quad balanceOf \circ ix\ (\text{JarHolder}\ id_{\texttt{jar}}) -= \texttt{wad}_0
\end{array}
$$

4.8 Manipulation

$\texttt{warp}\ t =$
$\quad \texttt{auth}\ \$\ \textbf{do}$
$\quad\quad \texttt{era} \mathrel{+}= t$

$\texttt{mine}\ id_{\texttt{jar}} = \textbf{do}$
$\quad \texttt{vat} \circ \texttt{jar}s \circ at\ id_{\texttt{jar}}\ ?= \texttt{Jar}\ \{$
$\quad\quad jarGem = \texttt{Gem}\ \{$
$\quad\quad\quad gemTotalSupply = 1000000000000,$
$\quad\quad\quad gemBalanceOf \ \ = singleton\ (\text{AddressHolder}\ id_{toy})\ 1000000000000$
$\quad\quad \},$
$\quad\quad jarTag = \texttt{Wad}\ 0,$
$\quad\quad jarZzz \ = 0$
$\quad \}$

$\texttt{hand}\ dst\ w\ id_{\texttt{jar}} = \textbf{do}$
$\quad g \ \leftarrow look\ (\texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem})$
$\quad g' \leftarrow transferFrom\ (\text{AddressHolder}\ id_{toy})\ (\text{AddressHolder}\ dst)\ w\ g$
$\quad \texttt{vat} \circ \texttt{jar}s \circ ix\ id_{\texttt{jar}} \circ \texttt{gem} := g'$

$\texttt{sire}\ \texttt{lad} = \textbf{do}$
$\quad accounts\ \%= (\texttt{lad}:)$

4.9 Other stuff

$perform :: \text{Act} \to \text{Maker}\ ()$
$perform\ x =$
$\quad \textbf{let}\ ?act = x\ \textbf{in case}\ x\ \textbf{of}$
$\quad\quad \texttt{Form}\ id\ \texttt{jar} \quad\ \to \texttt{form}\ id\ \texttt{jar}$
$\quad\quad \texttt{Mark}\ \texttt{jar}\ \texttt{tag}\ \texttt{zzz} \to \texttt{mark}\ \texttt{jar}\ \texttt{tag}\ \texttt{zzz}$
$\quad\quad \texttt{Open}\ id\ \texttt{ilk} \quad\ \to \texttt{open}\ id\ \texttt{ilk}$
$\quad\quad \texttt{Tell}\ \texttt{wad} \quad\quad \to \texttt{tell}\ \texttt{wad}$

```
    Frob ray      → frob ray
    Prod          → prod
    Warp t        → warp t
    Give urn lad → give urn lad
    Pull jar lad wad → pull jar lad wad
    Lock urn wad → lock urn wad
    Mine id       → mine id
    Hand lad wad jar → hand lad wad jar
    Sire lad      → sire lad
be :: Address → Act → Maker ()
be who x = do
  old ← use sender
  sender := who
  y ← perform x
  sender := old
  return y
transferFrom
  ::  (?act :: Act, MonadError Error m)
  ⇒  Holder → Holder → Wad
  →  Gem → m Gem
transferFrom src dst wad gem =
  case view (balanceOf ∘ at src) gem of
    Nothing →
      throwError (AssertError ?act)
    Just balance → do
      aver (balance ⩾ wad)
      return $ gem &~ do
        balanceOf ∘ ix src −= wad
        balanceOf ∘ at dst %=
          (λcase
            Nothing → Just wad
            Just x   → Just (wad + x))
```

Chapter 5

Act framework

5.1 Act descriptions

We define the Maker act vocabulary as a data type.

```
data Act =
    Bite (Id Urn)
  | Draw (Id Urn) Wad
  | Form (Id Ilk) (Id Jar)
  | Free (Id Urn) Wad
  | Frob Ray
  | Give (Id Urn) Address
  | Grab (Id Urn)
  | Heal Wad
  | Lock (Id Urn) Wad
  | Love Wad
  | Mark (Id Jar) Wad     Sec
  | Open (Id Urn) (Id Ilk)
  | Prod
  | Pull (Id Jar) Address Wad
  | Shut (Id Urn)
  | Tell Wad
  | Warp Sec
  | Wipe (Id Urn) Wad
  | Mine (Id Jar)
  | Hand Address Wad      (Id Jar)
  | Sire Address
```

```
Test acts
  | Addr Address
  deriving (Eq, Show)
```

Acts can fail. We divide the failure modes into general assertion failures and authentication failures.

```
data Error = AssertError Act | AuthError
  deriving (Show, Eq)
```

5.2 The Maker monad

The reader does not need any abstract understanding of monads to understand the code. What they give us is a nice syntax—the **do** notation—for expressing exceptions and state in a way that is still purely functional.

```
newtype Maker a =
  Maker (StateT System (Except Error) a)
  deriving
    (Functor, Applicative, Monad,
     MonadError Error,
     MonadState System)
```

```
exec :: System
     → Maker ()
     → Either Error System
exec sys (Maker m) =
  runExcept (execStateT m sys)
```

The following instance makes the mutable state also available as read-only state.

5.3 Accessor aliases

```
ilkAt  id = vat ∘ ilks ∘ ix id
urnAt id = vat ∘ urns ∘ ix id
jarAt  id = vat ∘ jars ∘ ix id
```

5.4 Asserting

$\texttt{aver}\ x = \mathit{unless}\ x\ (\mathit{throwError}\ (\mathrm{AssertError}\ ?\mathit{act}))$

$\mathit{look}\ f = \mathit{preuse}\ f \ggg\!= \lambda\mathbf{case}$
$\quad \mathrm{Nothing} \to \mathit{throwError}\ (\mathrm{AssertError}\ ?\mathit{act})$
$\quad \mathrm{Just}\ x \to \mathit{return}\ x$

5.5 Modifiers

$\texttt{auth}\ \mathit{continue} = \mathbf{do}$
$\quad s \leftarrow \mathit{use}\ \mathit{sender}$
$\quad \mathit{unless}\ (s \equiv \mathit{id}_{\mathit{god}})$
$\qquad (\mathit{throwError}\ \mathrm{AuthError})$
$\quad \mathit{continue}$

Chapter 6

Testing

Sketches for property stuff...

$$
\begin{aligned}
&\textbf{data}\ \text{Parameter} = \\
&\quad \texttt{Fix} \mid \texttt{Par} \mid \texttt{Way}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{maintains} \\
&\quad :: \text{Eq } a \Rightarrow \text{Lens}' \text{ System } a \to \text{Maker } () \\
&\qquad\qquad \to \text{System} \to \text{Bool} \\
&\mathit{maintains}\ p = \lambda m\ \texttt{sys}_0 \to \\
&\quad \textbf{case}\ \mathit{exec}\ \texttt{sys}_0\ m\ \textbf{of} \\
&\qquad \textsf{On success, data must be compared for equality} \\
&\qquad \text{Right } \texttt{sys}_1 \to \mathit{view}\ p\ \texttt{sys}_0 \equiv \mathit{view}\ p\ \texttt{sys}_1 \\
&\qquad \textsf{On rollback, data is maintained by definition} \\
&\qquad \text{Left } \_ \qquad \to \text{True}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{changesOnly} \\
&\quad ::\ \ \text{Lens}' \text{ System } a \to \text{Maker } () \\
&\quad \to \text{System} \to \text{Bool} \\
&\mathit{changesOnly}\ p = \lambda m\ \texttt{sys}_0 \to \\
&\quad \textbf{case}\ \mathit{exec}\ \texttt{sys}_0\ m\ \textbf{of} \\
&\qquad \textsf{On success, equalize } p \textsf{ and compare} \\
&\qquad \text{Right } \texttt{sys}_1 \to \mathit{set}\ p\ (\mathit{view}\ p\ \texttt{sys}_1)\ \texttt{sys}_0 \equiv \texttt{sys}_1 \\
&\qquad \textsf{On rollback, data is maintained by definition} \\
&\qquad \text{Left } \_ \qquad \to \text{True}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{also} :: \text{Lens}'\ s\ a \to \text{Lens}'\ s\ b \to \text{Lens}'\ s\ (a, b) \\
&\mathit{also}\ f\ g = \mathit{lens\ getter\ setter}
\end{aligned}
$$

where
$getter\ x = (view\ f\ x, view\ g\ x)$
$setter\ x\ (a, b) = set\ f\ a\ (set\ g\ b\ x)$

$keeps :: \text{Parameter} \to \text{Maker}\ () \to \text{System} \to \text{Bool}$
$keeps\ \texttt{Fix} = maintains\ (\texttt{vat} \circ \texttt{fix})$
$keeps\ \texttt{Par} = maintains\ (\texttt{vat} \circ \texttt{par})$
$keeps\ \texttt{Way} = maintains\ (\texttt{vat} \circ \texttt{way})$

Thus:

$foo\ \texttt{sys}_0 = all\ (\lambda f \to f\ \texttt{sys}_0)$
$\quad [\, changesOnly\ ((\texttt{vat} \circ \texttt{par})\ \text{`}also\text{`}$
$\qquad\qquad (\texttt{vat} \circ \texttt{way}))$
$\qquad (perform\ \texttt{Prod})]$

Appendix A

Prelude

```
module Maker.Prelude (
  module Maker.Prelude,
  module X
) where

import Prelude as X (
  -- Conversions to and from strings
  Read (..), Show (..),
  -- Comparisons
  Eq (..), Ord (..),
  -- Core abstractions
  Functor     (fmap),
  Applicative (),
  Monad       (return, (>>=)),
  -- Numeric classes
  Num (..), Integral (), Enum (),
  -- Numeric conversions
  Real (..), Fractional (..),
  RealFrac (..),
  fromIntegral,
  -- Simple types
  Integer, Int, String,
  -- Algebraic types
  Bool    (True, False),
```

$\text{Maybe}\ (\text{Just}, \text{Nothing}),$
$\text{Either}\ \ (\text{Right}, \text{Left}),$

Functional operators

$(\circ), (\,\$\,),$

Numeric operators

$(+), (-), (*), (/), (\uparrow), (\uparrow\uparrow), div,$

Utilities

$all,$

Constants

$mempty, \bot, otherwise)$

We use a typical composition of monad transformers from the `mtl` library to structure stateful actions. See section 5.2 (*The Maker monad*).

import Control.Monad.State *as* X (

MonadState,	Type class of monads with state
StateT,	Type constructor that adds state to a monad type
execStateT,	Runs a state monad with given initial state
get,	Gets the state in a **do** block
put)	Sets the state in a **do** block

import Control.Monad.Reader *as* X (

MonadReader,	Type class of monads with "environments"
ask,	Reads the environment in a **do** block
local)	Runs a sub-computation with a modified environment

import Control.Monad.Writer *as* X (

MonadWriter,	Type class of monads that emit logs
WriterT,	Type constructor that adds logging to a monad type
Writer,	Type constructor of logging monads
runWriterT,	Runs a writer monad transformer
execWriterT,	Runs a writer monad transformer keeping only logs
execWriter)	Runs a writer monad keeping only logs

import Control.Monad.Except *as* X (

MonadError,	Type class of monads that fail
Except,	Type constructor of failing monads
throwError,	Short-circuits the monadic computation
runExcept)	Runs a failing monad

Our numeric types use decimal fixed-point arithmetic.

import Data.Fixed *as* X (

Fixed (..),	Type constructor for numbers of given precision
HasResolution (..))	Type class for specifying precisions

We rely on the `lens` library for accessing nested values. There is no need to understand the theory behind lenses to understand this program. The notation $a \circ b \circ c$ denotes a nested accessor much like `a.b.c` in C-style languages; for more details, consult lens documentation[1].

import Control.Lens *as* X (

Lens′,	
lens,	
makeFields,	Defines lenses for record fields
set,	Writes a lens
use, *preuse*,	
zoom,	
view, *preview*,	Reads a lens in a **do** block
(&~),	Lets us use a **do** block with setters ◊ *Get rid of this.*
ix,	Lens for map retrieval and updating
at,	Lens for map insertion

Operators for partial state updates in **do** blocks:

(:=),	Replace
(−=), (+=),	Update arithmetically
(%=),	Update according to function
(?=))	Insert into map

Where the Solidity code uses `mapping`, we use Haskell's regular tree-based map type[2].

import Data.Map *as* X (

Map,	Type constructor for mappings
∅,	Polymorphic empty mapping
singleton)	Creates a mapping with a single key–value pair

For sequences of log entries, we use a sequence structure which has better time complexity than regular lists.

import Data.Sequence *as* X (Seq)
import *qualified* Data.Sequence *as* Sequence

Some less interesting imports are omitted from this document.

[1] Gabriel Gonzalez's 2013 article *Program imperatively using Haskell* is a good introduction.
[2] We assume the axiom that Keccak hash collisions are impossible.

Appendix B

Rounding fixed point numbers

This somewhat arcane-looking code implements a wrapper around the base library's decimal fixed point type, only with $x * y$ and $x \mathbin{/} y$ operations that do rounding instead of truncation of their intermediate results.

```
module Maker.Decimal (Decimal, E18, E36, Epsilon (..)) where

import Data.Fixed

newtype HasResolution e ⇒ Decimal e = D (Fixed e)
  deriving (Ord, Eq, Real, RealFrac)
```

We want the printed representations of these numbers to look like `"0.01"` and not `"R 0.01"`.

```
instance HasResolution e ⇒ Read (Decimal e) where
  readsPrec n s = fmap (λ(x, y) → (D x, y)) (readsPrec n s)
instance HasResolution e ⇒ Show (Decimal e) where
  show (D x) = show x
```

In the Num instance, we delegate everything except multiplication.

```
instance HasResolution e ⇒ Num (Decimal e) where
  x@(D (MkFixed a)) ∗ D (MkFixed b) =
    D (MkFixed (div (a ∗ b + div (resolution x) 2)
                     (resolution x)))
  D a + D b      = D (a + b)
  D a − D b      = D (a − b)
  negate  (D a) = D (negate a)
  abs     (D a) = D (abs a)
```

$$
\begin{aligned}
&\textit{signum}\ (\mathrm{D}\ a) = \mathrm{D}\ (\textit{signum}\ a)\\
&\textit{fromInteger}\ i = \mathrm{D}\ (\textit{fromInteger}\ i)
\end{aligned}
$$

In the Fractional instance, we delegate everything except division.

$$
\begin{aligned}
&\textbf{instance}\ \mathrm{HasResolution}\ e \Rightarrow \mathrm{Fractional}\ (\mathrm{Decimal}\ e)\ \textbf{where}\\
&\quad x@(\mathrm{D}\ (\mathrm{MkFixed}\ a))\ /\ \mathrm{D}\ (\mathrm{MkFixed}\ b) =\\
&\qquad \mathrm{D}\ (\mathrm{MkFixed}\ (\textit{div}\ (a * \textit{resolution}\ x + \textit{div}\ b\ 2)\ b))\\
&\quad \textit{recip}\ (\mathrm{D}\ a) \qquad = \mathrm{D}\ (\textit{recip}\ a)\\
&\quad \textit{fromRational}\ r = \mathrm{D}\ (\textit{fromRational}\ r)
\end{aligned}
$$

We define the E18 and E36 symbols and their fixed point multipliers.

$$
\begin{aligned}
&\textbf{data}\ \mathrm{E18}; \textbf{data}\ \mathrm{E36}\\
&\textbf{instance}\ \mathrm{HasResolution}\ \mathrm{E18}\ \textbf{where}\\
&\quad \textit{resolution}\ \_ = 10 \uparrow (18 :: \mathrm{Integer})\\
&\textbf{instance}\ \mathrm{HasResolution}\ \mathrm{E36}\ \textbf{where}\\
&\quad \textit{resolution}\ \_ = 10 \uparrow (36 :: \mathrm{Integer})
\end{aligned}
$$

The fixed point number types have well-defined smallest increments (denoted ϵ). This becomes useful when verifying equivalences.

$$
\begin{aligned}
&\textbf{class}\ \mathrm{Epsilon}\ t\ \textbf{where}\ \epsilon :: t\\
&\textbf{instance}\ \mathrm{HasResolution}\ a \Rightarrow \mathrm{Epsilon}\ (\mathrm{Decimal}\ a)\ \textbf{where}
\end{aligned}
$$

The use of $\perp$ is safe since *resolution* ignores the value.

$$
\epsilon = 1\ /\ \textit{fromIntegral}\ (\textit{resolution}\ (\perp :: \mathrm{Fixed}\ a))
$$