presents the

REFERENCE IMPLEMENTATION

of the remarkable

DAI CREDIT SYSTEM

issuing a diversely collateralized stablecoin

with last update on March 8, 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.

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

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

$$\textbf{newtype}\ \texttt{Wad} = \texttt{Wad}\ (\text{Fixed E18})$$
$$\quad \textbf{deriving}\ (\text{Ord}, \text{Eq}, \text{Num}, \text{Real}, \text{Fractional}, \text{RealFrac})$$

Define the distinct `ray` type for precise rate quantities

$$\textbf{newtype}\ \texttt{Ray} = \texttt{Ray}\ (\text{Fixed E36})$$
$$\quad \textbf{deriving}\ (\text{Ord}, \text{Eq}, \text{Num}, \text{Real}, \text{Fractional}, \text{RealFrac})$$

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

$$\textbf{data}\ \text{E18}; \textbf{data}\ \text{E36}$$

$$\textbf{instance}\ \text{HasResolution E18}\ \textbf{where}$$
$$\quad \mathit{resolution}\ \_ = 10 \uparrow (18 :: \text{Integer})$$
$$\textbf{instance}\ \text{HasResolution E36}\ \textbf{where}$$
$$\quad \mathit{resolution}\ \_ = 10 \uparrow (36 :: \text{Integer})$$

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.

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

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

$$\textbf{newtype}\ \texttt{Sec} = \texttt{Sec}\ \text{Int}$$
$$\quad \textbf{deriving}\ (\text{Eq}, \text{Ord}, \text{Enum}, \text{Num}, \text{Real}, \text{Integral})$$

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.

$$\textbf{data}\ \text{Id}\ a = \text{Id String}$$
$$\quad \textbf{deriving}\ (\text{Show}, \text{Eq}, \text{Ord})$$

We define another type for representing Ethereum account addresses.

$$\textbf{data}\ \text{Address} = \text{Address String}$$
$$\quad \textbf{deriving}\ (\text{Ord}, \text{Eq}, \text{Show})$$

We also have three predefined entity identifiers.

The DAI token address

$$id_{\text{DAI}} = \text{Id}\ \texttt{"Dai"}$$

The CDP engine address

$$id_{\texttt{vat}} = \text{Address}\ \texttt{"Vat"}$$

The account with ultimate authority
◊ *Kludge until authority is modelled*

$$id_{god} = \text{Address}\ \texttt{"God"}$$

This section introduces the records stored by the Maker system.

3.3 `Gem` — ERC20 token model

$$\textbf{data}\ \texttt{Gem} = \texttt{Gem}\ \{$$
$$\quad \mathit{gemTotalSupply} :: \texttt{Wad},$$
$$\quad \mathit{gemBalanceOf} :: \text{Map Address}\ \texttt{Wad},$$
$$\quad \mathit{gemAllowance} :: \text{Map (Address, Address)}\ \texttt{Wad}$$
$$\quad \}\ \textbf{deriving}\ (\text{Eq}, \text{Show})$$

3.4 Jar — collateral type

$$
\begin{array}{l}
\textbf{data}\ \texttt{Jar} = \texttt{Jar}\ \{ \\
\quad \begin{array}{lll}
\mathit{jarGem} & :: \texttt{Gem}, & \textsf{Collateral token} \\
\mathit{jarTag} & :: \texttt{Wad}, & \textsf{Market price} \\
\mathit{jarZzz} & :: \texttt{Sec} & \textsf{Price expiration}
\end{array} \\
\quad \}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Show})
\end{array}
$$

3.5 Ilk — CDP type

$$
\begin{array}{l}
\textbf{data}\ \texttt{Ilk} = \texttt{Ilk}\ \{ \\
\quad \begin{array}{lll}
\mathit{ilkJar} & :: \mathrm{Id}\ \texttt{Jar}, & \textsf{Collateral vault} \\
\mathit{ilkAxe} & :: \texttt{Ray}, & \textsf{Liquidation penalty} \\
\mathit{ilkHat} & :: \texttt{Wad}, & \textsf{Debt ceiling} \\
\mathit{ilkMat} & :: \texttt{Ray}, & \textsf{Liquidation ratio} \\
\mathit{ilkTax} & :: \texttt{Ray}, & \textsf{Stability fee} \\
\mathit{ilkLag} & :: \texttt{Sec}, & \textsf{Limbo duration} \\
\mathit{ilkRho} & :: \texttt{Sec}, & \textsf{Last dripped} \\
\mathit{ilkDin} & :: \texttt{Wad}, & \textsf{Total debt in dai} \\
\mathit{ilkChi} & :: \texttt{Ray} & \textsf{Debt unit}
\end{array} \\
\quad \}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Show})
\end{array}
$$

3.6 Urn — collateralized debt position (CDP)

$$
\begin{array}{l}
\textbf{data}\ \texttt{Urn} = \texttt{Urn}\ \{ \\
\quad \begin{array}{lll}
\mathit{urnCat} & :: \mathrm{Maybe\ Address}, & \textsf{Address of biting cat} \\
\mathit{urnVow} & :: \mathrm{Maybe\ Address}, & \textsf{Address of liquidating vow} \\
\mathit{urnLad} & :: \mathrm{Address}, & \textsf{Issuer} \\
\mathit{urnIlk} & :: \mathrm{Id}\ \texttt{Ilk}, & \textsf{CDP type} \\
\mathit{urnArt} & :: \texttt{Wad}, & \textsf{Outstanding debt in debt units} \\
\mathit{urnJam} & :: \texttt{Wad} & \textsf{Collateral amount in debt units}
\end{array} \\
\quad \}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Show})
\end{array}
$$

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
vatPie	:: 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

$$
\begin{aligned}
&\mathit{defaultIlk} :: \mathrm{Id}\ \mathtt{Jar} \to \mathtt{Ilk} \\
&\mathit{defaultIlk}\ \mathit{id}_{\mathtt{jar}} = \mathtt{Ilk}\ \{ \\
&\quad \mathit{ilkJar} \ = \mathit{id}_{\mathtt{jar}}, \\
&\quad \mathit{ilkAxe} = \mathtt{Ray}\ 1, \\
&\quad \mathit{ilkMat} = \mathtt{Ray}\ 1, \\
&\quad \mathit{ilkTax} = \mathtt{Ray}\ 1, \\
&\quad \mathit{ilkHat} = \mathtt{Wad}\ 0, \\
&\quad \mathit{ilkLag} = \mathtt{Sec}\ 0, \\
&\quad \mathit{ilkChi} = \mathtt{Ray}\ 1, \\
&\quad \mathit{ilkDin} = \mathtt{Wad}\ 0, \\
&\quad \mathit{ilkRho} = \mathtt{Sec}\ 0 \\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{defaultUrn} :: \mathrm{Id}\ \mathtt{Ilk} \to \mathrm{Address} \to \mathtt{Urn} \\
&\mathit{defaultUrn}\ \mathit{id}_{\mathtt{ilk}}\ \mathit{id}_{\mathtt{lad}} = \mathtt{Urn}\ \{ \\
&\quad \mathit{urnVow} = \mathrm{Nothing}, \\
&\quad \mathit{urnCat} \ = \mathrm{Nothing}, \\
&\quad \mathit{urnLad} \ = \mathit{id}_{\mathtt{lad}}, \\
&\quad \mathit{urnIlk} \ \ = \mathit{id}_{\mathtt{ilk}}, \\
&\quad \mathit{urnArt} \ = \mathtt{Wad}\ 0, \\
&\quad \mathit{urnJam} = \mathtt{Wad}\ 0 \\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{initialVat} :: \mathtt{Ray} \to \mathtt{Vat} \\
&\mathit{initialVat}\ \mathtt{how}_0 = \mathtt{Vat}\ \{ \\
&\quad \mathit{vatTau} \ = 0, \\
&\quad \mathit{vatFix} \ = \mathtt{Wad}\ 1, \\
&\quad \mathit{vatPar} \ = \mathtt{Wad}\ 1, \\
&\quad \mathit{vatHow} = \mathtt{how}_0, \\
&\quad \mathit{vatWay} = \mathtt{Ray}\ 1, \\
&\quad \mathit{vatPie} \ = \mathtt{Wad}\ 0, \\
&\quad \mathit{vatSin} \ = \mathtt{Wad}\ 0, \\
&\quad \mathit{vatIlks} \ = \varnothing, \\
&\quad \mathit{vatUrns} = \varnothing, \\
&\quad \mathit{vatJars} \ = \\
&\qquad \mathit{singleton}\ \mathit{id}_{\mathrm{DAI}}\ \mathtt{Jar}\ \{ \\
&\qquad\quad \mathit{jarGem} = \mathtt{Gem}\ \{
\end{aligned}
$$

$$
\begin{aligned}
&\qquad\quad \mathit{gemTotalSupply} = 0, \\
&\qquad\quad \mathit{gemBalanceOf} \;\; = \varnothing, \\
&\qquad\quad \mathit{gemAllowance} \;\; = \varnothing \\
&\qquad \}, \\
&\qquad \mathit{jarTag} = \texttt{Wad}\ 0, \\
&\qquad \mathit{jarZzz} \; = 0 \\
&\quad \} \\
&\}
\end{aligned}
$$

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

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

$id_{\texttt{ilk}} \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{ilk})$
$\texttt{drip}\ id_{\texttt{ilk}}$

Read parameters for risk analysis

$\texttt{era}_0 \leftarrow \mathit{view}\ \texttt{era}$
$\texttt{par}_0 \leftarrow \mathit{view}\ (\texttt{vat} \circ \texttt{par})$
$\texttt{urn}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\texttt{urn}})$
$\texttt{ilk}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \ (\mathit{view}\ \texttt{ilk}\ \texttt{urn}_0))$
$\texttt{jar}_0 \leftarrow \mathit{need}\ (\mathit{jarAt}\ \ (\mathit{view}\ \texttt{jar}\ \texttt{ilk}_0))$

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

$\texttt{open}\ id_{\texttt{urn}}\ id_{\texttt{ilk}} =$
$\quad \textbf{do}$
$\qquad id_{\texttt{lad}} \leftarrow \mathit{view}\ \mathit{sender}$
$\qquad \texttt{vat} \circ \texttt{urn}s \circ \mathit{at}\ id_{\texttt{urn}} \mathrel{?{=}} \mathit{defaultUrn}\ id_{\texttt{ilk}}\ id_{\texttt{lad}}$

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

Ensure CDP exists; identify collateral type

$$id_{\texttt{ilk}} \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{ilk})$$
$$id_{\texttt{jar}} \leftarrow need\ (ilkAt\quad id_{\texttt{ilk}} \circ \texttt{jar})$$

Record an increase in collateral

$$urnAt\ id_{\texttt{urn}} \circ \texttt{jam} \mathrel{+}= x$$

Take sender's tokens

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

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

Fail if sender is not the CDP owner.

$$id_{sender} \leftarrow view\ sender$$
$$id_{\texttt{lad}} \quad \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{lad})$$
$$\texttt{aver}\ (id_{sender} \equiv id_{\texttt{lad}})$$

Tentatively record the decreased collateral.

$$urnAt\ id_{\texttt{urn}} \circ \texttt{jam} \mathrel{-}= \texttt{wad}_{\texttt{gem}}$$

Fail if collateral decrease results in undercollateralization.

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

Send the collateral to the CDP owner.

$$id_{\texttt{ilk}} \leftarrow need\ (urnAt\quad id_{\texttt{urn}} \circ \texttt{ilk})$$
$$id_{\texttt{jar}} \leftarrow need\ (ilkAt\quad id_{\texttt{ilk}} \circ \texttt{jar})$$
$$\texttt{push}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ \texttt{wad}_{\texttt{gem}}$$

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

Fail if sender is not the CDP owner

$$id_{sender} \leftarrow view\ sender$$
$$id_{\texttt{lad}} \quad \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{lad})$$
$$\texttt{aver}\ (id_{sender} \equiv id_{\texttt{lad}})$$

Update price of debt coin

$$id_{\texttt{ilk}} \quad \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{ilk})$$
$$\texttt{chi}_1 \quad \leftarrow \texttt{drip}\ id_{\texttt{ilk}}$$

Denominate draw amount in debt coin

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

Increase debt

$$urnAt\ id_{\texttt{urn}} \circ \texttt{art} \mathrel{+}= \texttt{wad}_{\texttt{chi}}$$

Roll back unless overcollateralized

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

Mint dai and send to the CDP owner

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

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

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

Fail if sender is not the CDP owner

$id_{sender} \leftarrow view\ sender$

$id_{\texttt{lad}} \quad \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{lad})$

$\texttt{aver}\ (id_{sender} \equiv id_{\texttt{lad}})$

Update price of debt coin

$id_{\texttt{ilk}} \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{ilk})$

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

Denominate dai amount in debt coin

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

Roll back if the CDP is not overcollateralized

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

Reduce debt

$urnAt\ id_{\texttt{urn}} \circ \texttt{art} \mathrel{-}= \texttt{wad}_{\texttt{chi}}$

Take dai from CDP owner, or roll back

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

Destroy dai

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

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

$x \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{lad})$

$y \leftarrow view\ sender$

$\texttt{aver}\ (x \equiv y)$

$urnAt\ id_{\texttt{urn}} \circ \texttt{lad} := id_{\texttt{lad}}$

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

Update price of debt coin

$id_{\texttt{ilk}} \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{ilk})$

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

Attempt to repay all the CDP's outstanding dai

$\texttt{art}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ \mathit{id}_{\texttt{urn}} \circ \texttt{art})$

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

Reclaim all the collateral

$\mathit{jam0} \leftarrow \mathit{need}\ (\mathit{urnAt}\ \mathit{id}_{\texttt{urn}} \circ \texttt{jam})$

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

Nullify the CDP

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

4.3 Frequent adjustments

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

Time difference in seconds

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

Current target rate applied to target price

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

Sensitivity parameter applied over time

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

Target rate scaled up or down

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

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

Convert between multiplicative and additive form

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

$$\texttt{drip}\ \textit{id}_{\texttt{ilk}} = \textbf{do}$$

Current time stamp

$$
\begin{array}{l}
\quad \texttt{era}_0 \leftarrow \textit{view}\ \texttt{era} \\
\quad \texttt{rho}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ \textit{id}_{\texttt{ilk}} \circ \texttt{rho})
\end{array}
$$

Current stability fee

$$\quad \texttt{tax}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ \textit{id}_{\texttt{ilk}} \circ \texttt{tax})$$

Current price of debt coin

$$
\begin{array}{l}
\quad \texttt{chi}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ \textit{id}_{\texttt{ilk}} \circ \texttt{chi}) \\
\quad \textbf{let}
\end{array}
$$

$$\quad\quad \mathit{age} \;= \mathtt{era}_0 - \mathtt{rho}_0$$

$$\quad\quad \mathtt{chi}_1 = \mathtt{chi}_0 * \mathtt{tax}_0 \uparrow\uparrow \mathit{age}$$

$$\quad \mathit{ilkAt}\ \mathit{id}_{\mathtt{ilk}} \circ \mathtt{chi} := \mathtt{chi}_1$$

$$\quad \mathit{ilkAt}\ \mathit{id}_{\mathtt{ilk}} \circ \mathtt{rho} := \mathtt{era}_0$$

$$\quad \mathit{return}\ \mathtt{chi}_1$$

4.4 Governance

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

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

4.5 Price feedback

$$\mathtt{mark}\ \mathit{id}_{\mathtt{jar}}\ \mathtt{tag}_1\ \mathtt{zzz}_1 =$$
$$\quad \mathtt{auth}\ \$\ \mathbf{do}$$
$$\quad\quad \mathit{jarAt}\ \mathit{id}_{\mathtt{jar}} \circ \mathtt{tag} := \mathtt{tag}_1$$
$$\quad\quad \mathit{jarAt}\ \mathit{id}_{\mathtt{jar}} \circ \mathtt{zzz} := \mathtt{zzz}_1$$

$$\mathtt{tell}\ x =$$
$$\quad \mathtt{auth}\ \$\ \mathbf{do}$$
$$\quad\quad \mathtt{vat} \circ \mathtt{fix} := x$$

4.6 Liquidation and settlement

$$\mathtt{bite}\ \mathit{id}_{\mathtt{urn}} = \mathbf{do}$$

Fail if urn is not undercollateralized

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

Record the sender as the requester of liquidation

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

$\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{cat} := id_{\texttt{cat}}$

Read current debt

$\texttt{art}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{art})$

Update price of debt coin

$id_{\texttt{ilk}} \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{ilk})$

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

Read liquidation penalty ratio

$id_{\texttt{ilk}} \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{ilk})$

$\texttt{axe}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ id_{\texttt{ilk}} \circ \texttt{axe})$

Apply liquidation penalty to debt

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

Update debt and record it as in need of settlement

$\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{art} := \texttt{art}_1$

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

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

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

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

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

$\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{vow} := id_{\texttt{vow}}$

Clear the CDP's requester of liquidation

$\mathit{urnAt}\ id_{\texttt{urn}} \circ \texttt{cat} := \text{Nothing}$

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

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

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

$\texttt{loot}\ \texttt{wad}_{\text{DAI}} =$

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

$\quad \texttt{vat} \circ \texttt{pie} \mathrel{-}= \texttt{wad}_{\text{DAI}}$

4.7 Minting, burning, and transferring

$$
\begin{aligned}
&\texttt{pull}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ w = \textbf{do} \\
&\quad g \ \leftarrow need\ (jarAt\ id_{\texttt{jar}} \circ \texttt{gem}) \\
&\quad g' \leftarrow transferFrom\ id_{\texttt{lad}}\ id_{\texttt{vat}}\ w\ g \\
&\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} := g'
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{push}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ w = \textbf{do} \\
&\quad g \ \leftarrow need\ (jarAt\ id_{\texttt{jar}} \circ \texttt{gem}) \\
&\quad g' \leftarrow transferFrom\ id_{\texttt{vat}}\ id_{\texttt{lad}}\ w\ g \\
&\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} := g'
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{mint}\ id_{\texttt{jar}}\ \texttt{wad}_0 = \textbf{do} \\
&\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ totalSupply && \mathrel{+}= \texttt{wad}_0 \\
&\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ balanceOf \circ ix\ id_{\texttt{vat}} && \mathrel{+}= \texttt{wad}_0
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{burn}\ id_{\texttt{jar}}\ \texttt{wad}_0 = \textbf{do} \\
&\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ totalSupply && \mathrel{-}= \texttt{wad}_0 \\
&\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ balanceOf \circ ix\ id_{\texttt{vat}} && \mathrel{-}= \texttt{wad}_0
\end{aligned}
$$

4.8 Test-related manipulation

$$
\begin{aligned}
&\texttt{warp}\ t = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \texttt{era} \mathrel{+}= t
\end{aligned}
$$

4.9 Other stuff

```
perform :: Act → Maker ()
perform x =
  let ?act = x in case x of
    Form id jar    → form id jar
    Mark jar tag zzz → mark jar tag zzz
    Open id ilk    → open id ilk
    Tell wad       → tell 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
transferFrom
  ::  (MonadError Error m)
  ⇒  Address → Address → Wad
  →  Gem → m Gem
transferFrom src dst wad gem =
  case view (balanceOf ∘ at src) gem of
    Nothing →
      throwError AssertError
    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
  | Loot 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
  -- 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 | 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.

```
instance MonadReader System Maker where
  ask = Maker get
  local f (Maker m) = Maker $ do
    s ← get; put (f s)
    x ← m; put s
    return x
```

5.3 Constraints

```
type Reads r m = MonadReader r m
type Writes w m = MonadState w m
```

$$\textbf{type}\ \text{Fails}\quad m = \text{MonadError Error}\ m$$
$$\textbf{type}\ \text{IsAct} = ?act :: \text{Act}$$

5.4 Accessor aliases

$$\begin{aligned}
&ilkAt\ \ id = \texttt{vat} \circ \texttt{ilk}s \circ ix\ id\\
&urnAt\ id = \texttt{vat} \circ \texttt{urn}s \circ ix\ id\\
&jarAt\ \ id = \texttt{vat} \circ \texttt{jar}s \circ ix\ id
\end{aligned}$$

5.5 Asserting

$$\begin{aligned}
&\texttt{aver} :: \text{Fails}\ m \Rightarrow \text{Bool} \to m\ ()\\
&\texttt{aver}\ x = unless\ x\ (throwError\ \text{AssertError})\\
&need :: (\text{Fails}\ m, \text{Reads}\ r\ m)\\
&\quad \Rightarrow \text{Getting (First}\ a)\ r\ a \to m\ a\\
&need\ f = preview\ f \ggg\!= \lambda\textbf{case}\\
&\quad \text{Nothing} \to throwError\ \text{AssertError}\\
&\quad \text{Just}\ x \to return\ x
\end{aligned}$$

5.6 Modifiers

$$\begin{aligned}
&\texttt{auth} ::\\
&\quad (\text{IsAct}, \text{Fails}\ m,\\
&\quad\ \ \text{Reads}\ r\ m,\\
&\quad\quad \text{HasSender}\ r\ \text{Address})\\
&\quad \Rightarrow m\ a \to m\ a
\end{aligned}$$

$$\begin{aligned}
&\texttt{auth}\ continue = \textbf{do}\\
&\quad s \leftarrow view\ sender\\
&\quad unless\ (s \equiv id_{god})\\
&\quad\quad (throwError\ \text{AuthError})\\
&\quad continue
\end{aligned}$$

Chapter 6

Testing

Sketches for property stuff...

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

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

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

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

$$
\begin{array}{l}
\quad \textbf{where} \\
\quad \mathit{getter}\ x = (\mathit{view}\ f\ x, \mathit{view}\ g\ x) \\
\quad \mathit{setter}\ x\ (a, b) = \mathit{set}\ f\ a\ (\mathit{set}\ g\ b\ x)
\end{array}
$$

$$
\begin{array}{l}
\mathit{keeps} :: \text{Parameter} \rightarrow \text{Maker}\ () \rightarrow \text{System} \rightarrow \text{Bool} \\
\mathit{keeps}\ \texttt{Fix} = \mathit{maintains}\ (\texttt{vat} \circ \texttt{fix}) \\
\mathit{keeps}\ \texttt{Par} = \mathit{maintains}\ (\texttt{vat} \circ \texttt{par}) \\
\mathit{keeps}\ \texttt{Way} = \mathit{maintains}\ (\texttt{vat} \circ \texttt{way})
\end{array}
$$

Thus:

$$
\begin{array}{l}
\mathit{foo}\ \texttt{sys}_0 = \mathit{all}\ (\lambda f \rightarrow f\ \texttt{sys}_0) \\
\quad [\, \mathit{changesOnly}\ ((\texttt{vat} \circ \texttt{par})\ `\mathit{also}` \\
\qquad\qquad\qquad\qquad (\texttt{vat} \circ \texttt{way})) \\
\qquad (\mathit{perform}\ \texttt{Prod})\,]
\end{array}
$$

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 (toRational), Fractional (fromRational),
  RealFrac (truncate),
  fromIntegral,
 -- Simple types
  Integer, Int, String,
 -- Algebraic types
  Bool   (True, False),
```

Maybe (Just, Nothing),
Either (Right, Left),

Functional operators
$(\circ), (\,\$\,)$,

Numeric operators
$(+), (-), (*), (/), (\uparrow), (\uparrow\uparrow)$,

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
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

Act type signatures

$$
\begin{aligned}
&\textbf{type}\ \text{Numbers}\ \texttt{wad}\ \texttt{ray}\ \texttt{sec} = \\
&\quad (\texttt{wad}{\sim}\texttt{Wad}, \texttt{ray}{\sim}\texttt{Ray}, \texttt{sec}{\sim}\texttt{Sec})
\end{aligned}
$$

We see that `drip` may fail; it reads an `ilk`'s `tax`, `cow`, `rho`, and *bag*; and it writes those same parameters except `tax`.

$$
\begin{aligned}
&\texttt{drip} :: \\
&\quad (\text{Fails}\ m, \\
&\quad\ \ \text{Reads}\ r\ m, \\
&\quad\quad\quad \text{HasEra}\ r\ \texttt{Sec}, \\
&\quad\quad\quad \text{HasVat}\ r\ \texttt{vat}_r, \\
&\quad\quad\quad\quad \text{HasIlks}\ \texttt{vat}_r\ (\text{Map}\ (\text{Id}\ \texttt{Ilk})\ \texttt{ilk}_r), \\
&\quad\quad\quad\quad\quad \text{HasTax}\ \texttt{ilk}_r\ \texttt{Ray}, \\
&\quad\quad\quad\quad\quad \text{HasRho}\ \texttt{ilk}_r\ \texttt{Sec}, \\
&\quad\quad\quad\quad\quad \text{HasChi}\ \texttt{ilk}_r\ \texttt{Ray}, \\
&\quad\ \ \text{Writes}\ w\ m, \\
&\quad\quad\quad \text{HasVat}\ w\ \texttt{vat}_w, \\
&\quad\quad\quad\quad \text{HasIlks}\ \texttt{vat}_w\ (\text{Map}\ (\text{Id}\ \texttt{Ilk})\ \texttt{ilk}_w), \\
&\quad\quad\quad\quad\quad \text{HasRho}\ \texttt{ilk}_w\ \texttt{Sec}, \\
&\quad\quad\quad\quad\quad \text{HasChi}\ \texttt{ilk}_w\ \texttt{Ray}) \\
&\quad \Rightarrow \text{Id}\ \texttt{Ilk} \to m\ \texttt{Ray}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{form} :: \\
&\quad (\text{IsAct}, \text{Fails}\ m, \\
&\quad\ \ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ \text{Address}, \\
&\quad\ \ \text{Writes}\ w\ m, \text{HasVat}\ w\ \texttt{vat}_w,
\end{aligned}
$$

$$
\begin{aligned}
&\qquad\qquad\qquad\text{HasIlks}\ \texttt{vat}_w\ (\text{Map}\ (\text{Id}\ \texttt{Ilk})\ \texttt{Ilk}))\\
&\quad\Rightarrow \text{Id}\ \texttt{Ilk} \to \text{Id}\ \texttt{Jar} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{frob} :: (\text{IsAct}, \text{Fails}\ m,\\
&\qquad\quad \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ \text{Address},\\
&\qquad\quad \text{Writes}\ w\ m, \text{HasVat}\ w\ \texttt{vat}_w,\\
&\qquad\qquad\qquad\qquad \text{HasHow}\ \texttt{vat}_w\ \texttt{ray})\\
&\quad\Rightarrow \texttt{ray} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{open} ::\\
&\quad (\text{IsAct},\\
&\quad\ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ \text{Address},\\
&\quad\ \text{Writes}\ w\ m, \text{HasVat}\ w\ \texttt{vat}_w,\\
&\qquad\qquad\qquad \text{HasUrns}\ \texttt{vat}_w\ (\text{Map}\ (\text{Id}\ \texttt{Urn})\ \texttt{Urn}))\\
&\quad\Rightarrow \text{Id}\ \texttt{Urn} \to \text{Id}\ \texttt{Ilk} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{give} ::\\
&\quad (\text{IsAct}, \text{Fails}\ m,\\
&\quad\ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ \text{Address},\\
&\qquad\qquad\quad \text{HasVat}\ r\ \texttt{vat}_r,\\
&\qquad\qquad\qquad \text{HasUrns}\ \texttt{vat}_r\ (\text{Map}\ (\text{Id}\ \texttt{Urn})\ \texttt{urn}_r),\\
&\qquad\qquad\qquad\quad \text{HasLad}\ \texttt{urn}_r\ \text{Address},\\
&\quad\ \text{Writes}\ w\ m, \text{HasVat}\ w\ \texttt{vat}_r)\\
&\quad\Rightarrow \text{Id}\ \texttt{Urn} \to \text{Address} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{lock} ::\\
&\quad (\text{IsAct}, \text{Fails}\ m,\\
&\quad\ \text{Reads}\ r\ m,\\
&\qquad \text{HasSender}\ r\ \text{Address},\\
&\qquad \text{HasVat}\ r\ \texttt{vat}_r,\\
&\qquad\quad \text{HasUrns}\ \texttt{vat}_r\ (\text{Map}\ (\text{Id}\ \texttt{Urn})\ \texttt{urn}_r),\\
&\qquad\qquad \text{HasIlk}\ \texttt{urn}_r\ (\text{Id}\ \texttt{Ilk}),\\
&\qquad\quad \text{HasIlks}\ \texttt{vat}_r\ (\text{Map}\ (\text{Id}\ \texttt{Ilk})\ \texttt{ilk}_r),\\
&\qquad\qquad \text{HasJar}\ \texttt{ilk}_r\ (\text{Id}\ \texttt{Jar}),\\
&\qquad\quad \text{HasJars}\ \texttt{vat}_r\ (\text{Map}\ (\text{Id}\ \texttt{Jar})\ \texttt{jar}_r),\\
&\qquad\qquad \text{HasGem}\ \texttt{jar}_r\ \texttt{Gem},\\
&\quad\ \text{Writes}\ w\ m,\\
&\qquad \text{HasVat}\ w\ \texttt{vat}_w,
\end{aligned}
$$

$$
\begin{aligned}
&\quad\quad\quad \text{HasJars } \texttt{vat}_w \text{ (Map (Id } \texttt{Jar}\text{) } \texttt{jar}_r\text{)},\\
&\quad\quad\quad \text{HasUrns } \texttt{vat}_w \text{ (Map (Id } \texttt{Urn}\text{) } \texttt{urn}_w\text{)},\\
&\quad\quad\quad\quad \text{HasJam } \texttt{urn}_w \text{ } \texttt{Wad}\text{)}\\
&\quad \Rightarrow \text{Id } \texttt{Urn} \rightarrow \texttt{Wad} \rightarrow m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{mark} ::\\
&\quad (\text{IsAct}, \text{Fails } m,\\
&\quad\ \text{Reads } r\ m,\ \ \text{HasSender } r \text{ Address},\\
&\quad\ \text{Writes } w\ m, \text{HasVat } w\ \texttt{vat}_w,\\
&\quad\quad\quad\quad\quad\quad \text{HasJars } \texttt{vat}_w \text{ (Map (Id } \texttt{Jar}\text{) } \texttt{jar}_w\text{)},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasTag } \texttt{jar}_w\ \texttt{wad},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasZzz } \texttt{jar}_w\ \texttt{sec})\\
&\quad \Rightarrow \text{Id } \texttt{Jar} \rightarrow \texttt{wad} \rightarrow \texttt{sec} \rightarrow m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{tell} ::\\
&\quad (\text{IsAct}, \text{Fails } m,\\
&\quad\ \text{Reads } r\ m,\ \ \text{HasSender } r \text{ Address},\\
&\quad\ \text{Writes } w\ m, \text{HasVat } w\ \texttt{vat}_w,\\
&\quad\quad\quad\quad\quad\quad \text{HasFix } \texttt{vat}_w\ \texttt{wad})\\
&\quad \Rightarrow \texttt{wad} \rightarrow m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{prod} ::\\
&\quad (\text{IsAct},\\
&\quad\ \text{Reads } r\ m,\\
&\quad\quad \text{HasSender } r \text{ Address},\\
&\quad\quad \text{HasEra } r\ \texttt{sec},\\
&\quad\quad \text{HasVat } r\ \texttt{vat}_r,\ \ (\text{HasPar } \texttt{vat}_r\ \texttt{wad},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasTau } \texttt{vat}_r\ \texttt{sec},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasHow } \texttt{vat}_r\ \texttt{ray},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasWay } \texttt{vat}_r\ \texttt{ray},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasFix } \texttt{vat}_r\ \texttt{wad}),\\
&\quad\ \text{Writes } w\ m,\\
&\quad\quad \text{HasVat } w\ \texttt{vat}_w, (\text{HasPar } \texttt{vat}_w\ \texttt{wad},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasWay } \texttt{vat}_w\ \texttt{ray},\\
&\quad\quad\quad\quad\quad\quad\quad \text{HasTau } \texttt{vat}_w\ \texttt{sec}),\\
&\quad\ \text{Integral } \texttt{sec},\\
&\quad\ \text{Ord } \texttt{wad}, \text{Fractional } \texttt{wad},\\
&\quad\ \text{Fractional } \texttt{ray}, \text{Real } \texttt{ray})\\
&\quad \Rightarrow m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{warp} :: \\
&\quad (\mathrm{IsAct}, \mathrm{Fails}\ m, \\
&\quad\ \ \mathrm{Reads}\ r\ m,\ \ \mathrm{HasSender}\ r\ \mathrm{Address}, \\
&\quad\ \ \mathrm{Writes}\ w\ m, \mathrm{HasEra}\ w\ \texttt{sec}, \\
&\qquad\qquad\qquad\quad \mathrm{Num}\ \texttt{sec}) \\
&\quad \Rightarrow \texttt{sec} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{pull} :: \\
&\quad (\mathrm{Fails}\ m, \\
&\quad\ \ \mathrm{Reads}\ r\ m, \\
&\qquad \mathrm{HasVat}\ r\ \texttt{vat}_r,\ \ \mathrm{HasJars}\ \texttt{vat}_r\ (\mathrm{Map}\ (\mathrm{Id}\ \texttt{Jar})\ \texttt{jar}_r), \\
&\qquad\qquad\qquad\qquad\quad \mathrm{HasGem}\ \texttt{jar}_r\ \texttt{Gem}, \\
&\quad\ \ \mathrm{Writes}\ w\ m, \\
&\qquad \mathrm{HasVat}\ w\ \texttt{vat}_w, \mathrm{HasJars}\ \texttt{vat}_w\ (\mathrm{Map}\ (\mathrm{Id}\ \texttt{Jar})\ \texttt{jar}_r)) \\
&\quad \Rightarrow \mathrm{Id}\ \texttt{Jar} \to \mathrm{Address} \to \texttt{Wad} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{push} :: \\
&\quad (\mathrm{Fails}\ m, \\
&\quad\ \ \mathrm{Reads}\ r\ m, \\
&\qquad \mathrm{HasVat}\ r\ \texttt{vat}_r,\ \ \mathrm{HasJars}\ \texttt{vat}_r\ (\mathrm{Map}\ (\mathrm{Id}\ \texttt{Jar})\ \texttt{jar}_r), \\
&\qquad\qquad\qquad\qquad\quad \mathrm{HasGem}\ \texttt{jar}_r\ \texttt{Gem}, \\
&\quad\ \ \mathrm{Writes}\ w\ m, \\
&\qquad \mathrm{HasVat}\ w\ \texttt{vat}_w, \mathrm{HasJars}\ \texttt{vat}_w\ (\mathrm{Map}\ (\mathrm{Id}\ \texttt{Jar})\ \texttt{jar}_r)) \\
&\quad \Rightarrow \mathrm{Id}\ \texttt{Jar} \to \mathrm{Address} \to \texttt{Wad} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{mint} :: \\
&\quad (\mathrm{Fails}\ m, \\
&\quad\ \ \mathrm{Writes}\ w\ m, \\
&\qquad \mathrm{HasVat}\ w\ \texttt{vat}_w, \mathrm{HasJars}\ \texttt{vat}_w\ (\mathrm{Map}\ (\mathrm{Id}\ \texttt{Jar})\ \texttt{jar}_r), \\
&\qquad\qquad\qquad\qquad\quad \mathrm{HasGem}\ \texttt{jar}_r\ gem\_r, \\
&\qquad\qquad\qquad\qquad\qquad \mathrm{HasTotalSupply}\ gem\_r\ \texttt{Wad}, \\
&\qquad\qquad\qquad\qquad\qquad \mathrm{HasBalanceOf}\quad gem\_r\ (\mathrm{Map}\ \mathrm{Address}\ \texttt{Wad})) \\
&\quad \Rightarrow \mathrm{Id}\ \texttt{Jar} \to \texttt{Wad} \to m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{burn} :: \\
&\quad (\mathrm{Fails}\ m, \\
&\quad\ \ \mathrm{Writes}\ w\ m, \\
&\qquad \mathrm{HasVat}\ w\ \texttt{vat}_w, \mathrm{HasJars}\ \texttt{vat}_w\ (\mathrm{Map}\ (\mathrm{Id}\ \texttt{Jar})\ \texttt{jar}_r), \\
&\qquad\qquad\qquad\qquad\quad \mathrm{HasGem}\ \texttt{jar}_r\ gem\_r,
\end{aligned}
$$

$$
\begin{aligned}
&\qquad\qquad\qquad\qquad\quad \text{HasTotalSupply } \mathit{gem\_r}\ \texttt{Wad},\\
&\qquad\qquad\qquad\qquad\quad \text{HasBalanceOf } \quad \mathit{gem\_r}\ (\text{Map Address } \texttt{Wad}))\\
&\quad \Rightarrow \text{Id } \texttt{Jar} \rightarrow \texttt{Wad} \rightarrow m\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{grab} ::\\
&\quad (\text{IsAct}, \text{Fails } m,\\
&\quad\ \text{Numbers } \texttt{wad}\ \texttt{ray}\ \texttt{sec},\\
&\quad\ \text{Reads } r\ m,\\
&\quad\quad \text{HasSender } r \text{ Address},\\
&\quad\quad \text{HasEra } r\ \texttt{Sec},\\
&\quad\quad \text{HasVat } r\ \texttt{vat}_r,\\
&\quad\quad\quad \text{HasFix } \texttt{vat}_r\ \texttt{wad},\\
&\quad\quad\quad \text{HasPar } \texttt{vat}_r\ \texttt{wad},\\
&\quad\quad\quad \text{HasHow } \texttt{vat}_r\ \texttt{ray},\\
&\quad\quad\quad \text{HasWay } \texttt{vat}_r\ \texttt{ray},\\
&\quad\quad\quad \text{HasTau } \texttt{vat}_r\ \texttt{sec},\\
&\quad\quad\quad \text{HasUrns } \texttt{vat}_r\ (\text{Map (Id } \texttt{Urn})\ \texttt{urn}_r),\\
&\quad\quad\quad\quad \text{HasJam } \texttt{urn}_r\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasArt } \texttt{urn}_r\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasCat } \texttt{urn}_r\ (\text{Maybe Address}), \text{HasVow } \texttt{urn}_r\ (\text{Maybe Address}),\\
&\quad\quad\quad\quad \text{HasIlk } \texttt{urn}_r\ (\text{Id } \texttt{Ilk}),\\
&\quad\quad\quad \text{HasIlks } \texttt{vat}_r\ (\text{Map (Id } \texttt{Ilk})\ \texttt{ilk}_r),\\
&\quad\quad\quad\quad \text{HasHat } \texttt{ilk}_r\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasMat } \texttt{ilk}_r\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasDin } \texttt{ilk}_r\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasTax } \texttt{ilk}_r\ \texttt{ray},\\
&\quad\quad\quad\quad \text{HasLag } \texttt{ilk}_r\ \texttt{sec},\\
&\quad\quad\quad\quad \text{HasChi } \texttt{ilk}_r\ \texttt{ray},\ \ \text{HasRho } \texttt{ilk}_r\ \texttt{sec},\\
&\quad\quad\quad\quad \text{HasJar } \texttt{ilk}_r\ (\text{Id } \texttt{Jar}),\\
&\quad\quad\quad \text{HasJars } \texttt{vat}_r\ (\text{Map (Id } \texttt{Jar})\ \texttt{jar}_r),\\
&\quad\quad\quad\quad \text{HasGem } \texttt{jar}_r\ \texttt{Gem},\\
&\quad\quad\quad\quad \text{HasTag } \texttt{jar}_r\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasZzz } \texttt{jar}_r\ \texttt{sec},\\
&\quad\ \text{Writes } w\ m,\\
&\quad\quad \text{HasVat } w\ \texttt{vat}_w,\\
&\quad\quad\quad \text{HasTau } \texttt{vat}_w\ \texttt{sec},\\
&\quad\quad\quad \text{HasWay } \texttt{vat}_w\ \texttt{ray}, \text{HasPar } \texttt{vat}_w\ \texttt{wad},\\
&\quad\quad\quad \text{HasUrns } \texttt{vat}_w\ (\text{Map (Id } \texttt{Urn})\ \texttt{urn}_w),\\
&\quad\quad\quad\quad \text{HasJam } \texttt{urn}_w\ \texttt{wad}, \text{HasArt } \texttt{urn}_w\ \texttt{wad},\\
&\quad\quad\quad\quad \text{HasVow } \texttt{urn}_w \text{ Address},\\
&\quad\quad\quad\quad \text{HasCat } \texttt{urn}_w\ (\text{Maybe Address}),
\end{aligned}
$$

$$
\begin{aligned}
&\quad\quad \text{HasIlks}\ \texttt{vat}_w\ (\text{Map}\ (\text{Id}\ \texttt{Ilk})\ \texttt{ilk}_w), \\
&\quad\quad\quad \text{HasChi}\ \texttt{ilk}_w\ \texttt{ray}, \\
&\quad\quad\quad \text{HasRho}\ \texttt{ilk}_w\ \texttt{sec}, \\
&\quad\quad \text{HasJars}\ \texttt{vat}_w\ (\text{Map}\ (\text{Id}\ \texttt{Jar})\ \texttt{jar}_r) \\
&)\Rightarrow \text{Id}\ \texttt{Urn} \to m\ ()
\end{aligned}
$$