MAKER

*presents the*

REFERENCE IMPLEMENTATION

*of the remarkable*

# DAI CREDIT SYSTEM

issuing a diversely collateralized stablecoin

*with last update on February 28, 2017.*

# Contents

# 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 a previously unwritten mapping and get back a value initialized with zeroed memory, whereas in Haskell we must explicitly describe default values. The state rollback behavior of failed actions is also in Haskell explicitly coded as part of the monad transformer stack.

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.

## 1.2 Prerequisite Haskell knowledge

Some parts of this document require specific knowledge about Haskell programming, but these parts only make up a framework for expressing the more interesting parts in a natural way free of boilerplate.

◊ *Guidelines for skipping boring chapters and so on...*

For a complete understanding of the reference implementation's source code, the reader should grasp the following Haskell patterns:

- The use of **newtype** wrappers to distinguish different types of values which have the same underlying type.
- The use of **do** notation with the standard monad transformers:
  - `StateT` for updating state,
  - `ReaderT` for the read-only environment,
  - `WriterT` for "write-only state" (namely logs), and
  - `ExceptT` for failures which roll back state changes.
- The basic use of "lenses" (via the `lens` library) for convenient reading and writing of specific parts of nested values.
- The use of "parametricity" to express type-level guarantees about how function parameters are used, especially for understanding Appendix A which uses type signatures to specify which parts of the system are used or altered by each system action.
- ◊ *Some more stuff here...*

# Part I

# Implementation

# Chapter 2

# Preamble

We will begin by defining the program's basic dependencies before going on to define the basic data types and operations.

**module** Maker **where**

We use a typical stack of monad transformers from the `mtl` library to structure stateful actions; see section 4.2 (*The Maker monad*).

**import** Control.Monad.State
  (MonadState, StateT, *execStateT*, *get*, *put*)
**import** Control.Monad.Reader
  (MonadReader (..))
**import** Control.Monad.Writer
  (MonadWriter, WriterT, *runWriterT*)
**import** Control.Monad.Except
  (MonadError, Except, *throwError*, *runExcept*)

We use decimal fixed-point arithmetic.

**import** Data.Fixed (Fixed, HasResolution (..))

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 the lens manual.

**import** Control.Lens (

| | |
|---|---|
| $makeFields$, | Defines lenses for record fields |
| $view, preview$, | Reads a lens in a **do** block |
| $(\&\tilde{})$, | Lets us use a **do** block with setters ◊ *Get rid of this.* |
| $ix$, | Lens for map retrieval and updating |
| $at$, | Lens for map insertion |

Mutating operators for **do** blocks:

| | |
|---|---|
| $(:=)$, | Replace |
| $(-=), (+=), (*=)$, | Update arithmetically |
| $(\%=)$, | Update according to function |
| $(?=))$ | Insert into map |

Some less interesting imports are omitted from this document.

# Chapter 3

# Types

## 3.1 Numeric types

Many Ethereum tokens (e.g. ETH, DAI, and MKR) are denominated with 18 decimals. That makes decimal fixed point with 18 digits of precision a natural choice for representing currency quantities. We call such quantities "wads" (as in "wad of cash").

For some quantities, such as the rate of deflation per second, we want as much precision as possible, so we use twice the number of decimals. We call such quantities "rays" (mnemonic "rate," but also imagine a very precisely aimed ray of light).

Phantom types encode precision at compile time.

**data** E18; **data** E36

Specify $10^{-18}$ as the precision of E18.

**instance** HasResolution E18 **where**
  *resolution* _ = $10 \uparrow (18 :: \text{Integer})$

Specify $10^{-36}$ as the precision of E36.

**instance** HasResolution E36 **where**
  *resolution* _ = $10 \uparrow (36 :: \text{Integer})$

Create the distinct WAD type for currency quantities.

**newtype** WAD = WAD (Fixed E18)
  **deriving** (Ord, Eq, Num, Real, Fractional)

Create the distinct RAY type for precise rate quantities.

**newtype** RAY = RAY (Fixed E36)
  **deriving** (Ord, Eq, Num, Real, Fractional)

In calculations where a WAD is multiplied by a RAY, for example in the deflation mechanism, we have to downcast in a way that loses precision. Haskell does not cast automatically, so unless you see the following *cast* function applied, you can assume that precision is unchanged.

$$
\begin{aligned}
&\mathit{cast} :: (\text{Real } a, \text{Fractional } b) \Rightarrow a \to b \\
&\mathit{cast} = \\
&\quad \textsf{Convert via fractional } n/m \textsf{ form.} \\
&\quad \mathit{fromRational} \circ \mathit{toRational}
\end{aligned}
$$

We also define a type for non-negative integers.

$$
\begin{aligned}
&\textbf{newtype } \textsc{Nat} = \textsc{Nat} \text{ Int} \\
&\quad \textbf{deriving } (\text{Eq}, \text{Ord}, \text{Enum}, \text{Num}, \text{Real}, \text{Integral})
\end{aligned}
$$

### 3.1.1 Epsilon values

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

$$
\begin{aligned}
&\textbf{class } \text{Epsilon } t \textbf{ where } \epsilon :: t \\
&\textbf{instance } \text{HasResolution } a \Rightarrow \text{Epsilon } (\text{Fixed } a) \textbf{ where} \\
&\quad\quad \textsf{The use of } \bot \textsf{ is safe since } \mathit{resolution} \textsf{ ignores the value.} \\
&\quad \epsilon = 1 \mathbin{/} \mathit{fromIntegral}\ (\mathit{resolution}\ (\bot :: \text{Fixed } a)) \\
&\textbf{instance } \text{Epsilon } \textsc{Wad} \textbf{ where } \epsilon = \textsc{Wad}\ \epsilon \\
&\textbf{instance } \text{Epsilon } \textsc{Ray} \textbf{ where } \epsilon = \textsc{Ray}\ \epsilon
\end{aligned}
$$

## 3.2 Identifier type

There are several types of identifiers used in the system, and we can use Haskell's type system to distinguish them.

$$
\begin{aligned}
&\quad \textsf{The type parameter is only used to create distinct types.} \\
&\quad \textsf{For example, } \text{Id Foo} \textsf{ and } \text{Id Bar} \textsf{ are incompatible.} \\
&\textbf{data } \text{Id } a = \text{Id String} \\
&\quad \textbf{deriving } (\text{Show}, \text{Eq}, \text{Ord})
\end{aligned}
$$

It turns out that we will in several places use mappings from IDs to the value type corresponding to that ID type, so we define an alias for such mapping types.

$$\textbf{type}\ \mathrm{IdMap}\ a = \mathrm{Map}\ (\mathrm{Id}\ a)\ a$$

We also have three predefined entities:

The DAI token address

$$id_{\textsc{dai}} = \mathrm{Id}\ \texttt{"Dai"}$$

The CDP engine address

$$id_{\textsc{vat}} = \mathrm{Id}\ \texttt{"Vat"}$$

The account with ultimate authority

$$id_{god} = \mathrm{Id}\ \texttt{"God"}$$

## 3.3 Structures

[XXX: describe structures]

$$\textbf{data}\ \textsc{Lad} = \textsc{Lad}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Show})$$

### 3.3.1 Gem — Collateral token model

$$
\begin{array}{l}
\textbf{data}\ \textsc{Gem} = \\
\quad \textsc{Gem}\ \{ \\
\qquad \mathit{gemTotalSupply} :: !\textsc{Wad}, \\
\qquad \mathit{gemBalanceOf} :: !(\mathrm{Map}\ (\mathrm{Id}\ \textsc{Lad})\ \textsc{Wad}), \\
\qquad \mathit{gemAllowance} :: !(\mathrm{Map}\ (\mathrm{Id}\ \textsc{Lad}, \mathrm{Id}\ \textsc{Lad})\ \textsc{Wad}) \\
\quad \}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Read}, \mathrm{Show}) \\
\mathit{makeFields}\ \text{''}\ \textsc{Gem}
\end{array}
$$

### 3.3.2 Jar — Collateral token

$$\textbf{data}\ \textsc{Jar} = \textsc{Jar}\ \{$$

Collateral token

$jarGem :: !\text{GEM},$

Market price

$jarTag :: !\text{WAD},$

Price expiration

$jarZzz :: !\text{NAT}$

$\}$ **deriving** (Eq, Show, Read)

*makeFields* ' ' JAR

### 3.3.3 Ilk — cdp type

**data** ILK = ILK {

Collateral vault

$ilkJar :: !(\text{Id JAR}),$

Liquidation penalty

$ilkAxe :: !\text{RAY},$

Debt ceiling

$ilkHat :: !\text{WAD},$

Liquidation ratio

$ilkMat :: !\text{RAY},$

Stability fee

$ilkTax :: !\text{RAY},$

Limbo duration

$ilkLag :: !\text{NAT},$

Last dripped

$ilkRho :: !\text{NAT},$

???

$ilkCow :: !\text{RAY},$

Stability fee accumulator

$ilkBag :: !(\text{Map NAT RAY})$

$\}$ **deriving** (Eq, Show)

*makeFields* ' ' ILK

### 3.3.4 Urn — collateralized debt position (cdp)

**data** URN = URN {

Address of biting cat
$urnCat$ :: !(Maybe (Id LAD)),

Address of liquidating vow
$urnVow$ :: !(Maybe (Id LAD)),

Issuer
$urnLad$ :: !(Id LAD),

CDP type
$urnIlk$ :: !(Id ILK),

Outstanding dai debt
$urnCon$ :: !WAD,

Collateral amount
$urnPro$ :: !WAD,

Last poked
$urnPhi$ :: !NAT

} **deriving** (Eq, Show)

$makeFields$ '' URN

### 3.3.5 Vat — Dai creditor

**data** VAT = VAT {

Market price
$vatFix$ :: !WAD,

Sensitivity
$vatHow$ :: !RAY,

Target price
$vatPar$ :: !WAD,

Target rate
$vatWay$ :: !RAY,

Last prodded
$vatTau$ :: !NAT,

Unprocessed revenue from stability fees
$vatPie$ :: !WAD,

Bad debt from liquidated CDPs
$vatSin$ :: !WAD,

Collateral tokens

$$\begin{aligned}
&\quad \mathit{vatJars} :: !(\mathrm{IdMap}\ \textsc{Jar}),\\
&\quad \text{CDP types}\\
&\quad \mathit{vatIlks} :: !(\mathrm{IdMap}\ \textsc{Ilk}),\\
&\quad \text{CDPs}\\
&\quad \mathit{vatUrns} :: !(\mathrm{IdMap}\ \textsc{Urn})\\
&\}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Show})\\
&\mathit{makeFields}\ ''\ \textsc{Vat}
\end{aligned}$$

### 3.3.6 System model

$$\begin{aligned}
&\textbf{data}\ \mathrm{System} =\\
&\quad \mathrm{System}\ \{\\
&\qquad \mathit{systemVat} :: \textsc{Vat},\\
&\qquad \mathit{systemEra} :: !\textsc{Nat},\\
&\qquad \mathit{systemLads} :: \mathrm{IdMap}\ \textsc{Lad}, \quad \text{System users}\\
&\qquad \mathit{systemSender} :: \mathrm{Id}\ \textsc{Lad}\\
&\quad \}\ \textbf{deriving}\ (\mathrm{Eq}, \mathrm{Show})\\
&\mathit{makeFields}\ ''\ \mathrm{System}
\end{aligned}$$

### 3.3.7 Default data

$$\begin{aligned}
&\mathit{defaultIlk} :: \mathrm{Id}\ \textsc{Jar} \rightarrow \textsc{Ilk}\\
&\mathit{defaultIlk}\ \mathit{id}_{\textsc{jar}} = \textsc{Ilk}\ \{\\
&\quad \mathit{ilkJar} = \mathit{id}_{\textsc{jar}},\\
&\quad \mathit{ilkAxe} = \textsc{Ray}\ 1,\\
&\quad \mathit{ilkMat} = \textsc{Ray}\ 1,\\
&\quad \mathit{ilkTax} = \textsc{Ray}\ 1,\\
&\quad \mathit{ilkHat} = \textsc{Wad}\ 0,\\
&\quad \mathit{ilkLag} = \textsc{Nat}\ 0,\\
&\quad \mathit{ilkBag} = \varnothing,\\
&\quad \mathit{ilkCow} = \textsc{Ray}\ 1,\\
&\quad \mathit{ilkRho} = \textsc{Nat}\ 0\\
&\}
\end{aligned}$$

$$\begin{aligned}
&\mathit{defaultUrn} :: \mathrm{Id}\ \textsc{Ilk} \rightarrow \mathrm{Id}\ \textsc{Lad} \rightarrow \textsc{Urn}\\
&\mathit{defaultUrn}\ \mathit{id}_{\textsc{ilk}}\ \mathit{id}_{\textsc{lad}} = \textsc{Urn}\ \{
\end{aligned}$$

$$
\begin{aligned}
&\quad \mathit{urnVow} = \text{Nothing},\\
&\quad \mathit{urnCat} \;= \text{Nothing},\\
&\quad \mathit{urnLad} \;= \mathit{id}_{\text{LAD}},\\
&\quad \mathit{urnIlk} \;\;= \mathit{id}_{\text{ILK}},\\
&\quad \mathit{urnCon} = \text{WAD } 0,\\
&\quad \mathit{urnPro} \;= \text{WAD } 0,\\
&\quad \mathit{urnPhi} \;= \text{NAT } 0\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{initialVat} :: \text{RAY} \rightarrow \text{VAT}\\
&\mathit{initialVat}\ \text{HOW}_0 = \text{VAT}\ \{\\
&\quad \mathit{vatTau} \;= 0,\\
&\quad \mathit{vatFix} \;= \text{WAD } 1,\\
&\quad \mathit{vatPar} \;= \text{WAD } 1,\\
&\quad \mathit{vatHow} = \text{HOW}_0,\\
&\quad \mathit{vatWay} = \text{RAY } 1,\\
&\quad \mathit{vatPie} \;= \text{WAD } 0,\\
&\quad \mathit{vatSin} \;= \text{WAD } 0,\\
&\quad \mathit{vatIlks} \;= \varnothing,\\
&\quad \mathit{vatUrns} = \varnothing,\\
&\quad \mathit{vatJars} \;=\\
&\qquad \mathit{singleton}\ \mathit{id}_{\text{DAI}}\ \text{JAR}\ \{\\
&\qquad\quad \mathit{jarGem} = \text{GEM}\ \{\\
&\qquad\qquad \mathit{gemTotalSupply} = 0,\\
&\qquad\qquad \mathit{gemBalanceOf} \;\;= \varnothing,\\
&\qquad\qquad \mathit{gemAllowance} \;\;= \varnothing\\
&\qquad\quad \},\\
&\qquad\quad \mathit{jarTag} = \text{WAD } 0,\\
&\qquad\quad \mathit{jarZzz} \;= 0\\
&\qquad \}\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{initialSystem} :: \text{RAY} \rightarrow \text{System}\\
&\mathit{initialSystem}\ \text{HOW}_0 = \text{System}\ \{\\
&\quad \mathit{systemVat} \quad\;\; = \mathit{initialVat}\ \text{HOW}_0,\\
&\quad \mathit{systemLads} \quad = \varnothing,\\
&\quad \mathit{systemEra} \quad\;\; = 0,\\
&\quad \mathit{systemSender} = \mathit{id}_{god}\\
&\}
\end{aligned}
$$

# Chapter 4

# Act framework

## 4.1 Act descriptions

We define the Maker act vocabulary as a data type. This is used for logging and generally for representing acts.

```
data Act =
    Bite   (Id Urn)
  | Draw   (Id Urn) Wad
  | Form   (Id Ilk) (Id Jar)
  | Free   (Id Urn) Wad
  | Frob   Ray
  | Give   (Id Urn) (Id Lad)
  | Grab   (Id Urn)
  | Heal   Wad
  | Lock   (Id Urn) Wad
  | Loot   Wad
  | Mark   (Id Jar) Wad     Nat
  | Open   (Id Urn) (Id Ilk)
  | Prod
  | Poke   (Id Urn)
  | Pull   (Id Jar) (Id Lad) Wad
  | Shut   (Id Urn)
  | Tell   Wad
  | Warp   Nat
  | Wipe   (Id Urn) Wad
  | NewJar (Id Jar) Jar
```

```
    | NewLad (Id Lad)
    deriving (Eq, Show, Read)
```

Acts which are logged through the `note` modifier record the sender ID and the act descriptor.

```
data Log = LogNote (Id Lad) Act
  deriving (Show, Eq)
```

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

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

## 4.2 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, state, and logging in a way that is still purely functional.

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

exec :: System
     → Maker ()
     → Either Error (System, Seq Log)
exec sys (Maker m) =
  runExcept (runWriterT (execStateT m sys))
```

```
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
```

## 4.3 Constraints

```
type Reads  r m = MonadReader r m
type Writes w m = MonadState w m
type Logs     m = MonadWriter (Seq Log) m
type Fails    m = MonadError Error m

type IsAct = ?act :: Act
type Notes    m = (IsAct, Logs m)
```

## 4.4 Accessor aliases

```
ilkAt  id = VAT ∘ ILKs  ∘ ix id
urnAt  id = VAT ∘ URNs  ∘ ix id
jarAt  id = VAT ∘ JARs  ∘ ix id
```

## 4.5 Logging and asserting

```
log :: Logs m ⇒ Log → m ()
log x = Writer.tell (Sequence.singleton x)

sure :: Fails m ⇒ Bool → m ()
sure x = unless x (throwError AssertError)

need :: (Fails m, Reads r m)
  ⇒ Getting (First a) r a → m a
need f = preview f ≫= λcase
  Nothing → throwError AssertError
  Just x → return x
```

## 4.6 Modifiers

$$
\begin{array}{l}
\texttt{note} :: \\
\quad (\mathrm{IsAct}, \mathrm{Logs}\ m, \\
\quad\ \ \mathrm{Reads}\ r\ m, \\
\quad\quad \mathrm{HasSender}\ r\ (\mathrm{Id}\ \textsc{Lad})) \\
\quad \Rightarrow m\ a \rightarrow m\ a
\end{array}
$$

$$
\begin{array}{l}
\texttt{note}\ k = \mathbf{do} \\
\quad s \leftarrow \mathit{view\ sender} \\
\quad x \leftarrow k \\
\quad \mathit{log}\ (\mathrm{LogNote}\ s\ ?\mathit{act}) \\
\quad \mathit{return}\ x
\end{array}
$$

$$
\begin{array}{l}
\texttt{auth} :: \\
\quad (\mathrm{IsAct}, \mathrm{Fails}\ m, \\
\quad\ \ \mathrm{Reads}\ r\ m, \\
\quad\quad \mathrm{HasSender}\ r\ (\mathrm{Id}\ \textsc{Lad})) \\
\quad \Rightarrow m\ a \rightarrow m\ a
\end{array}
$$

$$
\begin{array}{l}
\texttt{auth}\ \mathit{continue} = \mathbf{do} \\
\quad s \leftarrow \mathit{view\ sender} \\
\quad \mathit{unless}\ (s \equiv \mathit{id}_{\mathit{god}}) \\
\quad\quad (\mathit{throwError}\ \mathrm{AuthError}) \\
\quad \mathit{continue}
\end{array}
$$

# Chapter 5

# Acts

We call the basic operations of the Dai credit system ”acts.”

Table 5.1: 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 |

## 5.1 Risk assessment

We divide an urn's situation into five stages of risk. Table 5.1 shows which acts each stage allows. The stages are naturally ordered from more to less risky.

```
data Stage = Dread | Grief | Panic | Worry | Anger | Pride
  deriving (Eq, Ord, Show)
```

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

```
analyze ERA₀ PAR₀ URN₀ ILK₀ JAR₀ =
  let
    Market value of collateral
     PRO_SDR = view PRO URN₀ * view TAG JAR₀
    Debt at DAI target price
     CON_SDR = view CON URN₀ * PAR₀
  in if
    Undergoing liquidation?
     | view VOW URN₀ ≢ Nothing                  → Dread
    Liquidation triggered?
     | view CAT  URN₀ ≢ Nothing                 → Grief
    Undercollateralized?
     | PRO_SDR < CON_SDR * view MAT ILK₀        → Panic
    Price feed expired?
     | ERA₀ > view ZZZ JAR₀ + view LAG ILK₀     → Panic
    Price feed in limbo?
     | view ZZZ  JAR₀ < ERA₀                    → Worry
    Debt ceiling reached?
     | view COW ILK₀  > view HAT ILK₀           → Anger
    Safely overcollateralized.
     | otherwise                                → Pride
```

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

$$
\begin{aligned}
&\texttt{gaze}\ id_{\text{URN}} = \textbf{do}\\
&\quad \texttt{prod}\\
&\quad \texttt{poke}\ id_{\text{URN}}\\
&\quad \text{ERA}_0 \leftarrow \mathit{view}\ \text{ERA}\\
&\quad \text{PAR}_0 \leftarrow \mathit{view}\ (\text{VAT} \circ \text{PAR})\\
&\quad \text{URN}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\text{URN}})\\
&\quad \text{ILK}_0 \ \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \ (\mathit{view}\ \text{ILK}\ \text{URN}_0))\\
&\quad \text{JAR}_0 \leftarrow \mathit{need}\ (\mathit{jarAt}\ \ (\mathit{view}\ \text{JAR}\ \text{ILK}_0\,))\\
&\quad \mathit{return}\ (\mathit{analyze}\ \text{ERA}_0\ \text{PAR}_0\ \text{URN}_0\ \text{ILK}_0\ \text{JAR}_0)
\end{aligned}
$$

## 5.2 Lending

$$
\begin{aligned}
&\texttt{open}\ id_{\text{URN}}\ id_{\text{ILK}} =\\
&\quad \texttt{note}\ \$\ \textbf{do}\\
&\qquad id_{\text{LAD}} \leftarrow \mathit{view}\ \mathit{sender}\\
&\qquad \text{VAT} \circ \text{URN}s \circ \mathit{at}\ id_{\text{URN}}\ ?\!= \mathit{defaultUrn}\ id_{\text{ILK}}\ id_{\text{LAD}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{lock}\ id_{\text{URN}}\ x =\\
&\quad \texttt{note}\ \$\ \textbf{do}\\
&\qquad \textsf{Ensure CDP exists; identify collateral type}\\
&\qquad id_{\text{ILK}} \ \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\text{URN}} \circ \text{ILK})\\
&\qquad id_{\text{JAR}} \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \ id_{\text{ILK}}\ \circ \text{JAR})\\
&\qquad \textsf{Record an increase in collateral}\\
&\qquad \mathit{urnAt}\ id_{\text{URN}} \circ \text{PRO} \mathrel{+}= x\\
&\qquad \textsf{Take sender's tokens}\\
&\qquad id_{\text{LAD}} \leftarrow \mathit{view}\ \mathit{sender}\\
&\qquad \texttt{pull}\ id_{\text{JAR}}\ id_{\text{LAD}}\ x
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{free}\ id_{\text{URN}}\ \text{WAD}_{\text{GEM}} =\\
&\quad \texttt{note}\ \$\ \textbf{do}
\end{aligned}
$$

Fail if sender is not the CDP owner.

$$\begin{aligned} &id_{sender} \leftarrow view\ sender \\ &id_{\mathrm{LAD}} \quad \leftarrow need\ (urnAt\ id_{\mathrm{URN}} \circ \mathrm{LAD}) \\ &sure\ (id_{sender} \equiv id_{\mathrm{LAD}}) \end{aligned}$$

Tentatively record the decreased collateral.

$$urnAt\ id_{\mathrm{URN}} \circ \mathrm{PRO} \mathrel{-=} \mathrm{WAD}_{\mathrm{GEM}}$$

Fail if collateral decrease results in undercollateralization.

$$\mathtt{gaze}\ id_{\mathrm{URN}} \ggg sure \circ (\equiv \mathtt{Pride})$$

Send the collateral to the CDP owner.

$$\begin{aligned} &id_{\mathrm{ILK}} \leftarrow need\ (urnAt \quad id_{\mathrm{URN}} \circ \mathrm{ILK}) \\ &id_{\mathrm{JAR}} \leftarrow need\ (ilkAt \quad id_{\mathrm{ILK}} \circ \mathrm{JAR}) \\ &\mathtt{push}\ id_{\mathrm{JAR}}\ id_{\mathrm{LAD}}\ \mathrm{WAD}_{\mathrm{GEM}} \end{aligned}$$

$$\mathtt{draw}\ id_{\mathrm{URN}}\ \mathrm{WAD}_{\mathrm{DAI}} =$$

$$\quad \mathtt{note}\ \$\ \mathbf{do}$$

Fail if sender is not the CDP owner.

$$\begin{aligned} &id_{sender} \leftarrow view\ sender \\ &id_{\mathrm{LAD}} \quad \leftarrow need\ (urnAt\ id_{\mathrm{URN}} \circ \mathrm{LAD}) \\ &sure\ (id_{sender} \equiv id_{\mathrm{LAD}}) \end{aligned}$$

Tentatively record DAI debt.

$$urnAt\ id_{\mathrm{URN}} \circ \mathrm{CON} \mathrel{+=} \mathrm{WAD}_{\mathrm{DAI}}$$

Fail if CDP with new debt is not overcollateralized.

$$\mathtt{gaze}\ id_{\mathrm{URN}} \ggg sure \circ (\equiv \mathtt{Pride})$$

Mint DAI and send it to the CDP owner.

$$\begin{aligned} &\mathtt{mint}\ id_{\mathrm{DAI}}\ \mathrm{WAD}_{\mathrm{DAI}} \\ &\mathtt{push}\ id_{\mathrm{DAI}}\ id_{\mathrm{LAD}}\ \mathrm{WAD}_{\mathrm{DAI}} \end{aligned}$$

$$\mathtt{wipe}\ id_{\mathrm{URN}}\ \mathrm{WAD}_{\mathrm{DAI}} =$$

$$\quad \mathtt{note}\ \$\ \mathbf{do}$$

Fail if sender is not the CDP owner.

$$\begin{aligned} &id_{sender} \leftarrow view\ sender \\ &id_{\mathrm{LAD}} \quad \leftarrow need\ (urnAt\ id_{\mathrm{URN}} \circ \mathrm{LAD}) \\ &sure\ (id_{sender} \equiv id_{\mathrm{LAD}}) \end{aligned}$$

Fail if the CDP is not currently overcollateralized.

$$\mathtt{gaze}\ id_{\mathrm{URN}} \ggg sure \circ (\equiv \mathtt{Pride})$$

Preliminarily reduce the CDP debt.

$urnAt\ id_{\text{URN}} \circ \text{CON} \mathrel{-{=}} \text{WAD}_{\text{DAI}}$

Attempt to get back DAI from CDP owner and destroy it.

$\texttt{pull}\ id_{\text{DAI}}\ id_{\text{LAD}}\ \text{WAD}_{\text{DAI}}$
$\texttt{burn}\ id_{\text{DAI}}\ \text{WAD}_{\text{DAI}}$

$\texttt{give}\ id_{\text{URN}}\ id_{\text{LAD}} =$
$\quad \texttt{note}\ \$\ \textbf{do}$
$\qquad x \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{LAD})$
$\qquad y \leftarrow view\ sender$
$\qquad sure\ (x \equiv y)$
$\qquad urnAt\ id_{\text{URN}} \circ \text{LAD} := id_{\text{LAD}}$

$\texttt{shut}\ id_{\text{URN}} =$
$\quad \texttt{note}\ \$\ \textbf{do}$

Update the CDP's debt (prorating the stability fee).

$\texttt{poke}\ id_{\text{URN}}$

Attempt to repay all the CDP's outstanding DAI.

$\text{CON}_0 \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{CON})$
$\texttt{wipe}\ id_{\text{URN}}\ \text{CON}_0$

Reclaim all the collateral.

$\text{PRO}_0 \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{PRO})$
$\texttt{free}\ id_{\text{URN}}\ \text{PRO}_0$

Nullify the CDP.

$\text{VAT} \circ \text{URN}s \circ at\ id_{\text{URN}} := \text{Nothing}$

## 5.3 Frequent adjustments

$$
\begin{aligned}
&\texttt{prod} = \texttt{note}\ \$\ \textbf{do}\\
&\quad \mathrm{ERA}_0 \leftarrow \mathit{view}\ \mathrm{ERA}\\
&\quad \mathrm{TAU}_0 \leftarrow \mathit{view}\ (\mathrm{VAT} \circ \mathrm{TAU})\\
&\quad \mathrm{FIX}_0 \leftarrow \mathit{view}\ (\mathrm{VAT} \circ \mathrm{FIX})\\
&\quad \mathrm{PAR}_0 \leftarrow \mathit{view}\ (\mathrm{VAT} \circ \mathrm{PAR})\\
&\quad \mathrm{HOW}_0 \leftarrow \mathit{view}\ (\mathrm{VAT} \circ \mathrm{HOW})\\
&\quad \mathrm{WAY}_0 \leftarrow \mathit{view}\ (\mathrm{VAT} \circ \mathrm{WAY})\\
&\quad \textbf{let}
\end{aligned}
$$

Time difference in seconds

$$\mathit{fan} = \mathrm{ERA}_0 - \mathrm{TAU}_0$$

Current deflation rate applied to target price

$$\mathrm{PAR}_1 = \mathrm{PAR}_0 * \mathit{cast}\ (\mathrm{WAY}_0 \uparrow\uparrow \mathit{fan})$$

Sensitivity parameter applied over time

$$\mathit{wag} = \mathrm{HOW}_0 * \mathit{fromIntegral}\ \mathit{fan}$$

Deflation rate scaled up or down

$$
\begin{aligned}
\mathrm{WAY}_1 = \mathit{inj}\ (&\mathit{prj}\ \mathrm{WAY}_0 +\\
&\textbf{if}\ \mathrm{FIX}_0 < \mathrm{PAR}_0\ \textbf{then}\ \mathit{wag}\ \textbf{else} - \mathit{wag})
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{VAT} \circ \mathrm{PAR} := \mathrm{PAR}_1\\
&\mathrm{VAT} \circ \mathrm{WAY} := \mathrm{WAY}_1\\
&\mathrm{VAT} \circ \mathrm{TAU} := \mathrm{ERA}_0\\
&\textbf{where}
\end{aligned}
$$

Convert between multiplicative and additive form

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

This internal act happens on every `poke`. It is also invoked when governance changes the TAX of an ILK.

$$
\begin{aligned}
&\texttt{drip}\ \mathit{id}_{\mathrm{ILK}} = \textbf{do}
\end{aligned}
$$

Current time stamp

$$\mathrm{ERA}_0 \leftarrow \mathit{view}\ \mathrm{ERA}$$

Current stability fee

$$
\begin{aligned}
&\mathrm{TAX}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\mathrm{ILK}} \circ \mathrm{TAX})\\
&\mathrm{COW}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\mathrm{ILK}} \circ \mathrm{COW})
\end{aligned}
$$

Previous time and stability fee thus far

$\text{RHO}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ id_{\text{ILK}} \circ \text{RHO})$
$\mathit{ice} \quad \leftarrow \mathit{need}\ (\mathit{ilkAt}\ id_{\text{ILK}} \circ \text{BAG} \circ \mathit{ix}\ \text{RHO}_0)$
**let**
Seconds passed
$\mathit{age} \quad = \text{ERA}_0 - \text{RHO}_0$
Stability fee accrued since last drip
$\mathit{dew} \quad = \mathit{ice} * \text{TAX}_0 \uparrow\uparrow \mathit{age}$
I don't understand this calculation
$\text{COW}_1 = \text{COW}_0 * (\mathit{dew} \,/\, \mathit{ice})$
$\mathit{ilkAt}\ id_{\text{ILK}} \circ \text{BAG} \circ \mathit{at}\ \text{ERA}_0 \mathrel{?=} \mathit{dew}$
$\mathit{ilkAt}\ id_{\text{ILK}} \circ \text{COW} \mathrel{:=} \text{COW}_1$
$\mathit{ilkAt}\ id_{\text{ILK}} \circ \text{RHO} \mathrel{:=} \text{ERA}_0$
$\mathit{return}\ \mathit{dew}$

`poke` $id_{\text{URN}} =$
`note` $ **do**
Read previous stability fee accumulator.
$id_{\text{ILK}} \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\text{URN}} \circ \text{ILK})$
$\mathit{phi0} \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\text{URN}} \circ \text{PHI})$
$\mathit{ice} \quad \leftarrow \mathit{need}\ (\mathit{ilkAt}\ id_{\text{ILK}} \circ \text{BAG} \circ \mathit{ix}\ \mathit{phi0})$
Update the stability fee accumulator.
$\text{CON}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ id_{\text{URN}} \circ \text{CON})$
$\mathit{dew} \quad \leftarrow$ `drip` $id_{\text{ILK}}$
Apply new stability fee to CDP debt.
$\mathit{urnAt}\ id_{\text{URN}} \circ \text{CON} \mathrel{*=} \mathit{cast}\ (\mathit{dew} \,/\, \mathit{ice})$
Record the poke time.
$\text{ERA}_0 \leftarrow \mathit{view}\ \text{ERA}$
$\mathit{urnAt}\ id_{\text{URN}} \circ \text{PHI} \mathrel{:=} \text{ERA}_0$

## 5.4 Governance

`form` $id_{\text{ILK}}\ id_{\text{JAR}} =$
`auth` $\circ$ `note` $ **do**
$\text{VAT} \circ \text{ILK}s \circ \mathit{at}\ id_{\text{ILK}} \mathrel{?=} \mathit{defaultIlk}\ id_{\text{JAR}}$

$$
\begin{aligned}
&\texttt{frob}\ \mathit{how}' = \\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
&\qquad \mathrm{VAT} \circ \mathrm{HOW} := \mathit{how}'
\end{aligned}
$$

## 5.5 Price feedback

$$
\begin{aligned}
&\texttt{mark}\ \mathit{id}_{\mathrm{JAR}}\ \mathrm{TAG}_1\ \mathrm{ZZZ}_1 = \\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
&\qquad \mathit{jarAt}\ \mathit{id}_{\mathrm{JAR}} \circ \mathrm{TAG} := \mathrm{TAG}_1 \\
&\qquad \mathit{jarAt}\ \mathit{id}_{\mathrm{JAR}} \circ \mathrm{ZZZ}\ \ := \mathrm{ZZZ}_1
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{tell}\ x = \\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
&\qquad \mathrm{VAT} \circ \mathrm{FIX} := x
\end{aligned}
$$

## 5.6 Liquidation and settlement

$\texttt{bite}\ \mathit{id}_{\mathrm{URN}} =$

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

Fail if urn is not undercollateralized.

$\texttt{gaze}\ \mathit{id}_{\mathrm{URN}} \ggg \mathit{sure} \circ (\equiv \texttt{Panic})$

Record the sender as the liquidation initiator.

$\mathit{id}_{\mathrm{CAT}} \leftarrow \mathit{view\ sender}$

$\mathit{urnAt}\ \mathit{id}_{\mathrm{URN}} \circ \mathrm{CAT} := \mathit{id}_{\mathrm{CAT}}$

Read current debt.

$\mathrm{CON}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ \mathit{id}_{\mathrm{URN}} \circ \mathrm{CON})$

Read liquidation penalty ratio.

$\mathit{id}_{\mathrm{ILK}}\ \leftarrow \mathit{need}\ (\mathit{urnAt}\ \mathit{id}_{\mathrm{URN}} \circ \mathrm{ILK})$

$\mathrm{AXE}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\mathrm{ILK}} \circ \mathrm{AXE})$

Apply liquidation penalty to debt.

$\textbf{let}\ \mathrm{CON}_1 = \mathrm{CON}_0 * \mathrm{AXE}_0$

Update debt and record it as in need of settlement.

$$\begin{aligned} &urnAt\ id_{\text{URN}} \circ \text{CON} := \text{CON}_1 \\ &\text{SIN} \qquad\qquad\qquad += \text{CON}_1 \end{aligned}$$

$\texttt{grab}\ id_{\text{URN}} =$

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

Fail if CDP liquidation is not initiated.

$\quad\texttt{gaze}\ id_{\text{URN}} \ggg sure \circ (\equiv \texttt{Grief})$

Record the sender as the CDP's settler.

$$\begin{aligned} &id_{\text{VOW}} \leftarrow view\ sender \\ &urnAt\ id_{\text{URN}} \circ \text{VOW} := id_{\text{VOW}} \end{aligned}$$

Nullify the CDP's debt and collateral.

$$\begin{aligned} &\text{PRO}_0 \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{PRO}) \\ &urnAt\ id_{\text{URN}} \circ \text{CON} := 0 \\ &urnAt\ id_{\text{URN}} \circ \text{PRO} := 0 \end{aligned}$$

Send the collateral to the settler for auctioning.

$$\begin{aligned} &id_{\text{ILK}} \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{ILK}) \\ &id_{\text{JAR}} \leftarrow need\ (ilkAt\ id_{\text{ILK}} \circ \text{JAR}) \\ &\texttt{push}\ id_{\text{JAR}}\ id_{\text{VOW}}\ \text{PRO}_0 \end{aligned}$$

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

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

$\qquad\text{VAT} \circ \text{SIN} -= \text{WAD}_{\text{DAI}}$

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

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

$\qquad\text{VAT} \circ \text{PIE} -= \text{WAD}_{\text{DAI}}$

## 5.7 Minting, burning, and transferring

$\texttt{pull}\ id_{\text{JAR}}\ id_{\text{LAD}}\ w = \textbf{do}$

$\quad g \leftarrow need\ (jarAt\ id_{\text{JAR}} \circ \text{GEM})$

$$
\begin{aligned}
&\quad g' \leftarrow \mathit{transferFrom}\ id_{\text{LAD}}\ id_{\text{VAT}}\ w\ g \\
&\quad \mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM} := g'
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{push}\ id_{\text{JAR}}\ id_{\text{LAD}}\ w = \textbf{do} \\
&\quad g\ \leftarrow \mathit{need}\ (\mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM}) \\
&\quad g' \leftarrow \mathit{transferFrom}\ id_{\text{VAT}}\ id_{\text{LAD}}\ w\ g \\
&\quad \mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM} := g'
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{mint}\ id_{\text{JAR}}\ \text{WAD}_0 = \textbf{do} \\
&\quad \mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM} \circ \mathit{totalSupply} \qquad\qquad \mathrel{+}= \text{WAD}_0 \\
&\quad \mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM} \circ \mathit{balanceOf} \circ \mathit{ix}\ id_{\text{VAT}} \mathrel{+}= \text{WAD}_0
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{burn}\ id_{\text{JAR}}\ \text{WAD}_0 = \textbf{do} \\
&\quad \mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM} \circ \mathit{totalSupply} \qquad\qquad \mathrel{-}= \text{WAD}_0 \\
&\quad \mathit{jarAt}\ id_{\text{JAR}} \circ \text{GEM} \circ \mathit{balanceOf} \circ \mathit{ix}\ id_{\text{VAT}} \mathrel{-}= \text{WAD}_0
\end{aligned}
$$

## 5.8 Test-related manipulation

$$
\begin{aligned}
&\texttt{warp}\ t = \\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
&\qquad \text{ERA} \mathrel{+}= t
\end{aligned}
$$

## 5.9 System modelling

$$\mathit{newLad}\ id_{\text{LAD}} = \mathit{lads} \circ \mathit{at}\ id_{\text{LAD}} \mathrel{?}= \textsc{Lad}$$

$$
\begin{aligned}
&\mathit{newLad} :: \\
&\quad (\text{Writes}\ w\ m, \text{HasLads}\ w\ (\text{IdMap}\ \textsc{Lad})) \\
&\quad \Rightarrow \text{Id}\ \textsc{Lad} \to m\ ()
\end{aligned}
$$

$$
\begin{array}{l}
\textit{newJar id id}_{\text{JAR}} = \\
\quad \texttt{auth} \circ \texttt{note} \ \$ \ \textbf{do} \\
\quad\quad \text{VAT} \circ \text{JAR}s \circ \textit{at id} \mathrel{?=} \textit{id}_{\text{JAR}}
\end{array}
$$

$$
\begin{array}{l}
\textit{newJar} :: \\
\quad (\ \text{IsAct}, \text{Fails}\ m, \text{Logs}\ m, \\
\quad\quad \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ (\text{Id}\ \text{LAD}), \\
\quad\quad \text{Writes}\ w\ m, \text{HasVat}\ w\ \text{VAT}_w, \\
\quad\quad\quad\quad\quad\quad \text{HasJars}\ \text{VAT}_w\ (\text{IdMap}\ \text{JAR})) \\
\quad \Rightarrow \\
\quad\quad \text{Id}\ \text{JAR} \to \text{JAR} \to m\ ()
\end{array}
$$

## 5.10 Other stuff

$$
\begin{array}{l}
\textit{perform} :: \text{Act} \to \text{Maker}\ () \\
\textit{perform}\ x = \\
\quad \textbf{let}\ ?act = x\ \textbf{in case}\ x\ \textbf{of} \\
\quad\quad \begin{array}{ll}
\text{NewLad}\ \textit{id} & \to \textit{newLad id} \\
\text{NewJar}\ \textit{id}\ \text{JAR} & \to \textit{newJar id}\ \text{JAR} \\
\texttt{Form}\ \textit{id}\ \text{JAR} & \to \texttt{form}\ \textit{id}\ \text{JAR} \\
\texttt{Mark}\ \text{JAR TAG ZZZ} & \to \texttt{mark}\ \text{JAR TAG ZZZ} \\
\texttt{Open}\ \textit{id}\ \text{ILK} & \to \texttt{open}\ \textit{id}\ \text{ILK} \\
\texttt{Tell}\ \text{WAD} & \to \texttt{tell}\ \text{WAD} \\
\texttt{Frob}\ \text{RAY} & \to \texttt{frob}\ \text{RAY} \\
\texttt{Prod} & \to \texttt{prod} \\
\texttt{Warp}\ t & \to \texttt{warp}\ t \\
\texttt{Give}\ \text{URN LAD} & \to \texttt{give}\ \text{URN LAD} \\
\texttt{Pull}\ \text{JAR LAD WAD} & \to \texttt{pull}\ \text{JAR LAD WAD} \\
\texttt{Lock}\ \text{URN WAD} & \to \texttt{lock}\ \text{URN WAD}
\end{array} \\
\textit{transferFrom} \\
\quad \begin{array}{ll}
:: & (\text{MonadError Error}\ m) \\
\Rightarrow & \text{Id}\ \text{LAD} \to \text{Id}\ \text{LAD} \to \text{WAD} \\
\to & \text{GEM} \to m\ \text{GEM}
\end{array} \\
\textit{transferFrom src dst}\ \text{WAD GEM} = \\
\quad \textbf{case}\ \textit{view}\ (\textit{balanceOf} \circ \textit{at src})\ \text{GEM}\ \textbf{of} \\
\quad\quad \text{Nothing} \to \\
\quad\quad\quad \textit{throwError}\ \text{AssertError}
\end{array}
$$

$$
\begin{array}{l}
\text{Just } \mathit{balance} \to \textbf{do} \\
\quad \mathit{sure}\ (\mathit{balance} \geqslant \text{WAD}) \\
\quad \mathit{return}\ \$\ \text{GEM} \mathbin{\&\tilde{}}\ \textbf{do} \\
\quad\quad \mathit{balanceOf} \circ \mathit{ix}\ \mathit{src} \mathrel{-{=}} \text{WAD} \\
\quad\quad \mathit{balanceOf} \circ \mathit{at}\ \mathit{dst} \mathrel{\%{=}} \\
\quad\quad\quad (\lambda\textbf{case} \\
\quad\quad\quad\quad \text{Nothing} \to \text{Just WAD} \\
\quad\quad\quad\quad \text{Just } x \quad\ \to \text{Just } (\text{WAD} + x))
\end{array}
$$

# Chapter 6

# Testing

# Appendix A

# Act type signatures

We see that `drip` may fail; it reads an ILK's TAX, COW, RHO, and BAG; and it writes those same parameters except TAX.

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

$$
\begin{aligned}
&\texttt{form} :: \\
&\quad (\text{IsAct}, \text{Fails } m, \text{Logs } m, \\
&\quad\ \ \text{Reads } r\ m,\ \ \text{HasSender } r\ (\text{Id } \textsc{Lad}), \\
&\quad\ \ \text{Writes } w\ m, \text{HasVat } w\ \textsc{vat}_w, \\
&\qquad\qquad\qquad\quad \text{HasIlks } \textsc{vat}_w\ (\text{IdMap } \textsc{Ilk}))
\end{aligned}
$$

$$\Rightarrow \text{Id ILK} \to \text{Id JAR} \to m\ ()$$

`frob` $:: (\text{IsAct}, \text{Fails } m, \text{Logs } m,$
$\quad\quad \text{Reads } r\ m,\ \ \text{HasSender } r\ (\text{Id LAD}),$
$\quad\quad \text{Writes } w\ m, \text{HasVat } w\ \text{VAT}_w,$
$\quad\quad\quad\quad\quad\quad \text{HasHow VAT}_w\ \text{RAY})$
$\quad \Rightarrow \text{RAY} \to m\ ()$

`open` $::$
$\quad (\text{IsAct}, \text{Logs } m,$
$\quad\ \text{Reads } r\ m,\ \ \text{HasSender } r\ (\text{Id LAD}),$
$\quad\ \text{Writes } w\ m, \text{HasVat } w\ \text{VAT}_w,$
$\quad\quad\quad\quad\quad \text{HasUrns VAT}_w\ (\text{IdMap URN}))$
$\quad \Rightarrow \text{Id URN} \to \text{Id ILK} \to m\ ()$

`give` $::$
$\quad (\text{IsAct}, \text{Fails } m, \text{Logs } m,$
$\quad\ \text{Reads } r\ m,\ \ \text{HasSender } r\ (\text{Id LAD}),$
$\quad\quad\quad\quad \text{HasVat } r\ \text{VAT}_r,$
$\quad\quad\quad\quad\quad \text{HasUrns VAT}_r\ (\text{Map (Id URN) URN}_r),$
$\quad\quad\quad\quad\quad\quad \text{HasLad URN}_r\ (\text{Id LAD}),$
$\quad\ \text{Writes } w\ m, \text{HasVat } w\ \text{VAT}_r)$
$\quad \Rightarrow \text{Id URN} \to \text{Id LAD} \to m\ ()$

`lock` $::$
$\quad (\text{IsAct}, \text{Fails } m, \text{Logs } m,$
$\quad\ \text{Reads } r\ m,$
$\quad\quad \text{HasSender } r\ (\text{Id LAD}),$
$\quad\quad \text{HasVat } r\ \text{VAT}_r,$
$\quad\quad\quad \text{HasUrns VAT}_r\ (\text{Map (Id URN) URN}_r),$
$\quad\quad\quad\quad \text{HasIlk URN}_r\ (\text{Id ILK}),$
$\quad\quad\quad \text{HasIlks VAT}_r\ (\text{Map (Id ILK) ILK}_r),$
$\quad\quad\quad\quad \text{HasJar ILK}_r\ (\text{Id JAR}),$
$\quad\quad\quad \text{HasJars VAT}_r\ (\text{Map (Id JAR) JAR}_r),$
$\quad\quad\quad\quad \text{HasGem JAR}_r\ \text{GEM},$
$\quad\ \text{Writes } w\ m,$
$\quad\quad \text{HasVat } w\ \text{VAT}_w,$
$\quad\quad\quad \text{HasJars VAT}_w\ (\text{Map (Id JAR) JAR}_r),$
$\quad\quad\quad \text{HasUrns VAT}_w\ (\text{Map (Id URN) URN}_w),$

$$
\begin{aligned}
&\qquad\qquad \text{HasPro } \textsc{urn}_w \ \textsc{Wad}) \\
&\quad \Rightarrow \text{Id } \textsc{Urn} \rightarrow \textsc{Wad} \rightarrow m \ ()
\end{aligned}
$$

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

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

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

$$
\begin{aligned}
&\texttt{warp} :: \\
&\quad (\text{IsAct}, \text{Fails } m, \text{Logs } m,
\end{aligned}
$$

Reads $r\ m$, HasSender $r$ (Id $\textsc{Lad}$),
Writes $w\ m$, HasEra $w$ $\textsc{nat}$,
Num $\textsc{nat}$)
$\Rightarrow \textsc{nat} \rightarrow m\ ()$

`pull` ::
(Fails $m$,
Reads $r\ m$,
HasVat $r$ $\textsc{vat}_r$, HasJars $\textsc{vat}_r$ (Map (Id $\textsc{Jar}$) $\textsc{jar}_r$),
HasGem $\textsc{jar}_r$ $\textsc{Gem}$,
Writes $w\ m$,
HasVat $w$ $\textsc{vat}_w$, HasJars $\textsc{vat}_w$ (Map (Id $\textsc{Jar}$) $\textsc{jar}_r$))
$\Rightarrow$ Id $\textsc{Jar} \rightarrow$ Id $\textsc{Lad} \rightarrow \textsc{Wad} \rightarrow m\ ()$

`push` ::
(Fails $m$,
Reads $r\ m$,
HasVat $r$ $\textsc{vat}_r$, HasJars $\textsc{vat}_r$ (Map (Id $\textsc{Jar}$) $\textsc{jar}_r$),
HasGem $\textsc{jar}_r$ $\textsc{Gem}$,
Writes $w\ m$,
HasVat $w$ $\textsc{vat}_w$, HasJars $\textsc{vat}_w$ (Map (Id $\textsc{Jar}$) $\textsc{jar}_r$))
$\Rightarrow$ Id $\textsc{Jar} \rightarrow$ Id $\textsc{Lad} \rightarrow \textsc{Wad} \rightarrow m\ ()$

`mint` ::
(Fails $m$,
Writes $w\ m$,
HasVat $w$ $\textsc{vat}_w$, HasJars $\textsc{vat}_w$ (Map (Id $\textsc{Jar}$) $\textsc{jar}_r$),
HasGem $\textsc{jar}_r$ $gem\_r$,
HasTotalSupply $gem\_r$ $\textsc{Wad}$,
HasBalanceOf $gem\_r$ (Map (Id $\textsc{Lad}$) $\textsc{Wad}$))
$\Rightarrow$ Id $\textsc{Jar} \rightarrow \textsc{Wad} \rightarrow m\ ()$

`burn` ::
(Fails $m$,
Writes $w\ m$,
HasVat $w$ $\textsc{vat}_w$, HasJars $\textsc{vat}_w$ (Map (Id $\textsc{Jar}$) $\textsc{jar}_r$),
HasGem $\textsc{jar}_r$ $gem\_r$,
HasTotalSupply $gem\_r$ $\textsc{Wad}$,
HasBalanceOf $gem\_r$ (Map (Id $\textsc{Lad}$) $\textsc{Wad}$))
$\Rightarrow$ Id $\textsc{Jar} \rightarrow \textsc{Wad} \rightarrow m\ ()$