MAKER

presents the

REFERENCE IMPLEMENTATION

of the remarkable

DAI CREDIT SYSTEM

issuing a diversely collateralized stablecoin

with last update on February 26, 2017.

Contents

Chapter 1

Introduction

The "Dai credit system" is the smart contract system used by the DAI MAKER to control the price stability and deflation of the DAI stablecoin by automatic modification of market incentives (via deflation adjustment), and to provide trustless credit services to Ethereum blockchain users.

New dai enter the money supply when a dai borrower posts an excess of collateral to a "collateralized debt position" (CDP) and takes out a loan. The debt and collateral amounts are recorded in the CDP, and (as time passes) the stability fees incurred by the CDP owner are also recorded. The collateral itself is held in a token vault controlled by the DAI MAKER.

Any Ethereum account can borrow dai without any requirements beyond posting and maintaining adequate collateral. There are no term limits on dai loans and borrowers are free to open or close CDPs at any time. The collateral held in CDPs collectively backs the value of the dai in a fully transparent manner that anyone can verify.

1.1 Motivation

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. The reasons for maintaining this "reference implementation" in Haskell are, roughly:

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.

Part I

Implementation

Chapter 2

Preamble

```
module Maker where
```

We import types for the decimal fixed-point arithmetic which we use for amounts and rates.

```
import Data.Fixed
```

We rely on the `lens` library for defining and using accessors which otherwise tend to become long-winded in Haskell. Since our program has several nested records, this makes the code much clearer. There is no need to understand the theory behind lenses to understand this program. All the reader needs to know is that $a \circ b \circ c$ denotes a nested accessor much like `a.b.c` in C-style languages. The rest should be obvious from context.

```
import Control.Lens
```

We use a typical stack of monad transformers from the *mtl* library to structure state-modifying actions. Again, the reader does not need any abstract understanding of monads. They make our code clear and simple by enabling **do** blocks to express exceptions, state, and logging.

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

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 ↑ (18 :: Integer)

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

instance HasResolution E36 **where**
  resolution _ = 10 ↑ (36 :: 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.

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

We also define a type for non-negative integers.

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

3.1.1 Epsilon values

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

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

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.

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

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.

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

We also have three predefined entities:

The DAI token address
$id_{\text{DAI}} = \text{Id}$ `"Dai"`

The CDP engine address
$id_{\text{VAT}} = \text{Id}$ `"Vat"`

The account with ultimate authority
$id_{god} = \text{Id}$ `"God"`

3.3 Structures

[XXX: describe structures]

```
data Lad = Lad deriving (Eq, Show)
```

3.3.1 Gem — Collateral token model

```
data Gem =
  Gem {
    gemTotalSupply :: !Wad,
    gemBalanceOf   :: !(Map (Id Lad)          Wad),
    gemAllowance   :: !(Map (Id Lad, Id Lad) Wad)
  } deriving (Eq, Read, Show)
makeFields ''Gem
```

3.3.2 Jar — Collateral token

```
data Jar = Jar {
    -- Collateral token
    jarGem :: !Gem,
    -- Market price
    jarTag :: !Wad,
    -- Price expiration
    jarZzz :: !Nat
  } deriving (Eq, Show, Read)
makeFields ''Jar
```

3.3.3 Ilk — CDP type

```
data Ilk = Ilk {
    -- Collateral vault
    ilkJar  :: !(Id Jar),
    -- Liquidation penalty
    ilkAxe  :: !Ray,
    -- Debt ceiling
    ilkHat  :: !Wad,
    -- Liquidation ratio
    ilkMat  :: !Ray,
    -- Stability fee
    ilkTax  :: !Ray,
    -- Limbo duration
    ilkLag  :: !Nat,
    -- Last dripped
    ilkRho  :: !Nat,
    -- ???
    ilkCow  :: !Ray,
    -- Stability fee accumulator
    ilkBag  :: !(Map Nat Ray)
  } deriving (Eq, Show)
makeFields ''Ilk
```

3.3.4 Urn — 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
$\quad urnCon :: !\textsc{Wad},$

Collateral amount
$\quad urnPro :: !\textsc{Wad},$

Last poked
$\quad urnPhi :: !\textsc{Nat}$

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

$makeFields$ ' ' $\textsc{Urn}$

3.3.5 Vat — Dai creditor

data $\textsc{Vat} = \textsc{Vat}\ \{$

Market price
$\quad vatFix :: !\textsc{Wad},$

Sensitivity
$\quad vatHow :: !\textsc{Ray},$

Target price
$\quad vatPar :: !\textsc{Wad},$

Target rate
$\quad vatWay :: !\textsc{Ray},$

Last prodded
$\quad vatTau :: !\textsc{Nat},$

Unprocessed revenue from stability fees
$\quad vatPie :: !\textsc{Wad},$

Bad debt from liquidated CDPs
$\quad vatSin :: !\textsc{Wad},$

Collateral tokens
$\quad vatJars :: !(\text{IdMap } \textsc{Jar}),$

CDP types
$\quad vatIlks :: !(\text{IdMap } \textsc{Ilk}),$

CDPs
$\quad vatUrns :: !(\text{IdMap } \textsc{Urn})$

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

$makeFields$ ' ' $\textsc{Vat}$

3.3.6 System model

$$
\begin{aligned}
&\textbf{data}\ \text{System} = \\
&\quad \text{System}\ \{ \\
&\qquad \mathit{systemVat} \quad :: \text{V{\scriptsize AT}}, \\
&\qquad \mathit{systemEra} \quad :: !\text{N{\scriptsize AT}}, \\
&\qquad \mathit{systemLads} \quad :: \text{IdMap L{\scriptsize AD}}, \quad \textsf{System users} \\
&\qquad \mathit{systemSender} :: \text{Id L{\scriptsize AD}} \\
&\quad \}\ \textbf{deriving}\ (\text{Eq}, \text{Show}) \\
&\mathit{makeFields}\ {}''\ \text{System}
\end{aligned}
$$

3.3.7 Default data

$$
\begin{aligned}
&\mathit{defaultIlk} :: \text{Id J{\scriptsize AR}} \to \text{I{\scriptsize LK}} \\
&\mathit{defaultIlk}\ \mathit{id}_{\text{JAR}} = \text{I{\scriptsize LK}}\ \{ \\
&\quad \mathit{ilkJar} \ = \mathit{id}_{\text{JAR}}, \\
&\quad \mathit{ilkAxe} = \text{R{\scriptsize AY}}\ 1, \\
&\quad \mathit{ilkMat} = \text{R{\scriptsize AY}}\ 1, \\
&\quad \mathit{ilkTax} = \text{R{\scriptsize AY}}\ 1, \\
&\quad \mathit{ilkHat} = \text{W{\scriptsize AD}}\ 0, \\
&\quad \mathit{ilkLag} = \text{N{\scriptsize AT}}\ 0, \\
&\quad \mathit{ilkBag} = \varnothing, \\
&\quad \mathit{ilkCow} = \text{R{\scriptsize AY}}\ 1, \\
&\quad \mathit{ilkRho} = \text{N{\scriptsize AT}}\ 0 \\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{defaultUrn} :: \text{Id I{\scriptsize LK}} \to \text{Id L{\scriptsize AD}} \to \text{U{\scriptsize RN}} \\
&\mathit{defaultUrn}\ \mathit{id}_{\text{ILK}}\ \mathit{id}_{\text{LAD}} = \text{U{\scriptsize RN}}\ \{ \\
&\quad \mathit{urnVow} = \text{Nothing}, \\
&\quad \mathit{urnCat} = \text{Nothing}, \\
&\quad \mathit{urnLad} = \mathit{id}_{\text{LAD}}, \\
&\quad \mathit{urnIlk} \ = \mathit{id}_{\text{ILK}}, \\
&\quad \mathit{urnCon} = \text{W{\scriptsize AD}}\ 0, \\
&\quad \mathit{urnPro} = \text{W{\scriptsize AD}}\ 0, \\
&\quad \mathit{urnPhi} = \text{N{\scriptsize AT}}\ 0 \\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{initialVat} :: \text{R{\scriptsize AY}} \to \text{V{\scriptsize AT}} \\
&\mathit{initialVat}\ \text{{\scriptsize HOW}}_0 = \text{V{\scriptsize AT}}\ \{
\end{aligned}
$$

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

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

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)
```

Now we can define the type of a

```
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
```

4.2 Constraints

$$
\begin{aligned}
&\textbf{type}\ \text{Reads}\ r\ m = \text{MonadReader}\ r\ m\\
&\textbf{type}\ \text{Writes}\ w\ m = \text{MonadState}\ w\ m\\
&\textbf{type}\ \text{Logs}\ m = \text{MonadWriter}\ (\text{Seq Log})\ m\\
&\textbf{type}\ \text{Fails}\ m = \text{MonadError Error}\ m\\
&\textbf{type}\ \text{IsAct} = ?act :: \text{Act}\\
&\textbf{type}\ \text{Notes}\ m = (\text{IsAct}, \text{Logs}\ m)
\end{aligned}
$$

4.3 Accessor aliases

$$
\begin{aligned}
&ilkAt\ id = \text{VAT} \circ \text{ILK}s \circ ix\ id\\
&urnAt\ id = \text{VAT} \circ \text{URN}s \circ ix\ id\\
&jarAt\ id = \text{VAT} \circ \text{JAR}s \circ ix\ id
\end{aligned}
$$

4.4 Logging and asserting

$$
\begin{aligned}
&log :: \text{Logs}\ m \Rightarrow \text{Log} \to m\ ()\\
&log\ x = \text{Writer.}\texttt{tell}\ (\text{Sequence.}singleton\ x)\\
&sure :: \text{Fails}\ m \Rightarrow \text{Bool} \to m\ ()\\
&sure\ x = unless\ x\ (throwError\ \text{AssertError})\\
&need :: (\text{Fails}\ m, \text{Reads}\ r\ m)\\
&\quad \Rightarrow \text{Getting}\ (\text{First}\ a)\ r\ a \to m\ a\\
&need\ f = preview\ f \gg\!= \lambda\textbf{case}\\
&\quad \text{Nothing} \to throwError\ \text{AssertError}\\
&\quad \text{Just}\ x \to return\ x
\end{aligned}
$$

4.5 Modifiers

4.5.1 `note` — logging actions

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

$$\begin{aligned}
&\quad \text{Reads}\ r\ m, \\
&\qquad \text{HasSender}\ r\ (\text{Id}\ \textsc{Lad})) \\
&\quad \Rightarrow m\ a \rightarrow m\ a
\end{aligned}$$

$$\begin{aligned}
&\texttt{note}\ k = \textbf{do} \\
&\quad s \leftarrow \mathit{view}\ \mathit{sender} \\
&\quad x \leftarrow k \\
&\quad \mathit{log}\ (\text{LogNote}\ s\ ?\mathit{act}) \\
&\quad \mathit{return}\ x
\end{aligned}$$

4.5.2 **`auth`** — authenticating actions

$$\begin{aligned}
&\texttt{auth} :: \\
&\quad (\text{IsAct}, \text{Fails}\ m, \\
&\quad\ \text{Reads}\ r\ m, \\
&\qquad \text{HasSender}\ r\ (\text{Id}\ \textsc{Lad})) \\
&\quad \Rightarrow m\ a \rightarrow m\ a
\end{aligned}$$

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

Chapter 5

Acts

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

Table 5.1: Possible acts in the stages of an urn

	`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 Acts performed by other acts

5.1.1 `gaze` — identify urn stage

The internal non-mutating act `gaze` identifies the *stage* of an URN. It is used to assert that other acts are invoked correctly according to table 5.1.

First we define the stages. Note that they are ordered from "bad to good," letting us say, for example, that `shut` id_{URN} is allowed when `gaze` $id_{\text{URN}} \geqslant$ `Panic`.

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

First we define the function *analyze* implementing the logic of urn stages.

$$
\begin{array}{l}
\mathit{analyze}\ \text{ERA}_0\ \text{PAR}_0\ \text{URN}_0\ \text{ILK}_0\ \text{JAR}_0 = \\
\quad \textbf{let} \\
\qquad \textsf{Locked collateral market price} \\
\qquad \text{PRO}_{\text{SDR}} = \mathit{view}\ \text{PRO}\ \text{URN}_0 * \mathit{view}\ \text{TAG}\ \text{JAR}_0 \\
\qquad \textsf{Debt at DAI target price} \\
\qquad \text{CON}_{\text{SDR}} = \mathit{view}\ \text{CON}\ \text{URN}_0 * \text{PAR}_0 \\
\qquad \textsf{Does debt-to-collateral ratio exceed liquidation ratio?} \\
\qquad \mathit{risky} = \text{CON}_{\text{SDR}} * \mathit{view}\ \text{MAT}\ \text{ILK}_0 > \text{PRO}_{\text{SDR}} \\
\qquad \textsf{Does price feed latency exceed limbo duration?} \\
\qquad \mathit{laggy} = \mathit{view}\ \text{ZZZ}\ \text{JAR}_0 + \mathit{view}\ \text{LAG}\ \text{ILK}_0 < \text{ERA}_0 \\
\quad \textbf{in if} \\
\qquad \mid \mathit{view}\ \text{VOW}\ \text{URN}_0 \not\equiv \text{Nothing} \to \texttt{Dread} \\
\qquad \mid \mathit{view}\ \text{CAT}\ \text{URN}_0 \not\equiv \text{Nothing} \to \texttt{Grief} \\
\qquad \mid \mathit{risky} \vee \mathit{laggy} \to \texttt{Panic} \\
\qquad \mid \mathit{view}\ \text{ZZZ}\ \text{JAR}_0 < \text{ERA}_0 \to \texttt{Worry} \\
\qquad \mid \mathit{view}\ \text{COW}\ \text{ILK}_0 > \mathit{view}\ \text{HAT}\ \text{ILK}_0 \to \texttt{Anger} \\
\qquad \mid \mathit{otherwise} \to \texttt{Pride}
\end{array}
$$

Now we define the `gaze` act which preliminarily updates the relevant parameters and then returns the value of *analyze*.

$$
\begin{array}{l}
\texttt{gaze}\ id_{\text{URN}} = \mathbf{do} \\
\quad \texttt{prod} \\
\quad \texttt{poke}\ id_{\text{URN}} \\
\quad \text{ERA}_0 \leftarrow \textit{view}\ \text{ERA} \\
\quad \text{PAR}_0 \leftarrow \textit{view}\ (\text{VAT} \circ \text{PAR}) \\
\quad \text{URN}_0 \leftarrow \textit{need}\ (\textit{urnAt}\ id_{\text{URN}}) \\
\quad \text{ILK}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ (\textit{view}\ \text{ILK}\ \text{URN}_0)) \\
\quad \text{JAR}_0 \leftarrow \textit{need}\ (\textit{jarAt}\ (\textit{view}\ \text{JAR}\ \text{ILK}_0)) \\
\quad \textit{return}\ (\textit{analyze}\ \text{ERA}_0\ \text{PAR}_0\ \text{URN}_0\ \text{ILK}_0\ \text{JAR}_0)
\end{array}
$$

5.1.2 `drip` — update stability fee accumulator

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

$$
\begin{array}{l}
\texttt{drip}\ id_{\text{ILK}} = \mathbf{do} \\
\quad \textsf{Current time stamp} \\
\quad \text{ERA}_0 \leftarrow \textit{view}\ \text{ERA} \\
\quad \textsf{Current stability fee} \\
\quad \text{TAX}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ id_{\text{ILK}} \circ \text{TAX}) \\
\quad \text{COW}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ id_{\text{ILK}} \circ \text{COW}) \\
\quad \textsf{Previous time and stability fee thus far} \\
\quad \text{RHO}_0 \leftarrow \textit{need}\ (\textit{ilkAt}\ id_{\text{ILK}} \circ \text{RHO}) \\
\quad \textit{ice} \quad\ \leftarrow \textit{need}\ (\textit{ilkAt}\ id_{\text{ILK}} \circ \text{BAG} \circ \textit{ix}\ \text{RHO}_0) \\
\quad \mathbf{let} \\
\qquad \textsf{Seconds passed} \\
\qquad \textit{age} = \text{ERA}_0 - \text{RHO}_0 \\
\qquad \textsf{Stability fee accrued since last drip} \\
\qquad \textit{dew} = \textit{ice} * \text{TAX}_0 \uparrow\uparrow \textit{age} \\
\qquad \textsf{I don't understand this calculation} \\
\qquad \text{COW}_1 = \text{COW}_0 * (\textit{dew}\ /\ \textit{ice}) \\
\quad \textit{ilkAt}\ id_{\text{ILK}} \circ \text{BAG} \circ \textit{at}\ \text{ERA}_0\ ?= \textit{dew} \\
\quad \textit{ilkAt}\ id_{\text{ILK}} \circ \text{COW} := \text{COW}_1 \\
\quad \textit{ilkAt}\ id_{\text{ILK}} \circ \text{RHO} := \text{ERA}_0 \\
\quad \textit{return}\ \textit{dew}
\end{array}
$$

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

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

5.2 Acts performed by governance

5.2.1 `form` — create a new ilk

$$
\begin{array}{l}
\texttt{form}\ id_{\textsc{ilk}}\ id_{\textsc{jar}} = \\
\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
\qquad \textsc{vat} \circ \textsc{ilk}s \circ \mathit{at}\ id_{\textsc{ilk}}\ ?\!= \mathit{defaultIlk}\ id_{\textsc{jar}}
\end{array}
$$

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

5.2.2 `frob` — alter the sensitivity parameter

$$
\begin{aligned}
&\texttt{frob}\ how' = \\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
&\qquad \textsc{vat} \circ \textsc{how} := how'
\end{aligned}
$$

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

5.3 Acts performed by account holders

5.3.1 `open` — open CDP

$$
\begin{aligned}
&\texttt{open}\ id_{\textsc{urn}}\ id_{\textsc{ilk}} = \\
&\quad \texttt{note}\ \$\ \textbf{do} \\
&\qquad id_{\textsc{lad}} \leftarrow view\ sender \\
&\qquad \textsc{vat} \circ \textsc{urn}s \circ at\ id_{\textsc{urn}}\ ?= defaultUrn\ id_{\textsc{ilk}}\ id_{\textsc{lad}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{open} :: \\
&\quad (\text{IsAct}, \text{Logs}\ m, \\
&\quad\ \ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ (\text{Id}\ \text{L}\textsc{ad}), \\
&\quad\ \ \text{Writes}\ w\ m, \text{HasVat}\ w\ \textsc{vat}_w, \\
&\qquad\qquad\qquad \text{HasUrns}\ \textsc{vat}_w\ (\text{IdMap}\ \text{U}\textsc{rn})) \\
&\quad \Rightarrow \text{Id}\ \text{U}\textsc{rn} \to \text{Id}\ \text{I}\textsc{lk} \to m\ ()
\end{aligned}
$$

5.3.2 `give` — transfer CDP

$$
\begin{aligned}
&\texttt{give}\ id_{\textsc{urn}}\ id_{\textsc{lad}} = \\
&\quad \texttt{note}\ \$\ \textbf{do} \\
&\qquad x \leftarrow need\ (urnAt\ id_{\textsc{urn}} \circ \textsc{lad}) \\
&\qquad y \leftarrow view\ sender \\
&\qquad sure\ (x \equiv y) \\
&\qquad urnAt\ id_{\textsc{urn}} \circ \textsc{lad} := id_{\textsc{lad}}
\end{aligned}
$$

$$
\begin{array}{l}
\texttt{give} :: \\
\quad (\text{IsAct}, \text{Fails}\ m, \text{Logs}\ m, \\
\quad\ \ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ (\text{Id}\ \text{L}\textsc{ad}), \\
\qquad\qquad\qquad\quad \text{HasVat}\ r\ \textsc{vat}_r, \\
\qquad\qquad\qquad\qquad \text{HasUrns}\ \textsc{vat}_r\ (\text{Map}\ (\text{Id}\ \text{U}\textsc{rn})\ \textsc{urn}_r), \\
\qquad\qquad\qquad\qquad\quad \text{HasLad}\ \textsc{urn}_r\ (\text{Id}\ \text{L}\textsc{ad}), \\
\quad\ \ \text{Writes}\ w\ m, \text{HasVat}\ w\ \textsc{vat}_r) \\
\quad\ \ \Rightarrow \text{Id}\ \text{U}\textsc{rn} \to \text{Id}\ \text{L}\textsc{ad} \to m\ ()
\end{array}
$$

5.3.3 **`shut`** — repay dai, reclaim collateral, and delete CDP

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

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

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

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

Attempt to repay all the CDP's outstanding DAI.

$\quad \text{CON}_0 \leftarrow \textit{need}\ (\textit{urnAt}\ id_{\text{URN}} \circ \text{CON})$

$\quad \texttt{wipe}\ id_{\text{URN}}\ \text{CON}_0$

Reclaim all the collateral.

$\quad \text{PRO}_0 \leftarrow \textit{need}\ (\textit{urnAt}\ id_{\text{URN}} \circ \text{PRO})$

$\quad \texttt{free}\ id_{\text{URN}}\ \text{PRO}_0$

Nullify the CDP.

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

5.3.4 **`lock`** — insert collateral

$\texttt{lock}\ id_{\text{URN}}\ x =$

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

Ensure CDP exists; identify collateral type

$\quad id_{\text{ILK}} \leftarrow \textit{need}\ (\textit{urnAt}\ id_{\text{URN}} \circ \text{ILK})$

$\quad id_{\text{JAR}} \leftarrow \textit{need}\ (\textit{ilkAt}\ \ id_{\text{ILK}} \circ \text{JAR})$

Record an increase in collateral

$\quad \textit{urnAt}\ id_{\text{URN}} \circ \text{PRO} \mathrel{+}= x$

Take sender's tokens

$\quad id_{\text{LAD}} \leftarrow \textit{view}\ \textit{sender}$

$\quad \texttt{pull}\ id_{\text{JAR}}\ id_{\text{LAD}}\ x$

$$
\begin{array}{l}
\texttt{lock} :: \\
\quad (\text{IsAct}, \text{Fails } m, \text{Logs } m, \\
\quad\ \text{Reads } r\ m, \\
\qquad \text{HasSender } r\ (\text{Id } \textsc{Lad}), \\
\qquad \text{HasVat } r\ \textsc{vat}_r, \\
\qquad\quad \text{HasUrns } \textsc{vat}_r\ (\text{Map } (\text{Id } \textsc{Urn})\ \textsc{urn}_r), \\
\qquad\qquad \text{HasIlk } \textsc{urn}_r\ (\text{Id } \textsc{Ilk}), \\
\qquad\quad \text{HasIlks } \textsc{vat}_r\ (\text{Map } (\text{Id } \textsc{Ilk})\ \textsc{ilk}_r), \\
\qquad\qquad \text{HasJar } \textsc{ilk}_r\ (\text{Id } \textsc{Jar}), \\
\qquad\quad \text{HasJars } \textsc{vat}_r\ (\text{Map } (\text{Id } \textsc{Jar})\ \textsc{jar}_r), \\
\qquad\qquad \text{HasGem } \textsc{jar}_r\ \textsc{Gem}, \\
\quad\ \text{Writes } w\ m, \\
\qquad \text{HasVat } w\ \textsc{vat}_w, \\
\qquad\quad \text{HasJars } \textsc{vat}_w\ (\text{Map } (\text{Id } \textsc{Jar})\ \textsc{jar}_r), \\
\qquad\quad \text{HasUrns } \textsc{vat}_w\ (\text{Map } (\text{Id } \textsc{Urn})\ \textsc{urn}_w), \\
\qquad\qquad \text{HasPro } \textsc{urn}_w\ \textsc{Wad}) \\
\quad \Rightarrow \text{Id } \textsc{Urn} \rightarrow \textsc{Wad} \rightarrow m\ ()
\end{array}
$$

5.3.5 **`wipe`** — pay back dai debt

$$
\begin{array}{l}
\texttt{wipe}\ id_{\text{URN}}\ \text{WAD}_{\text{DAI}} = \\
\quad \texttt{note}\ \$\ \textbf{do}
\end{array}
$$

Fail if sender is not the CDP owner.

$$
\begin{array}{l}
id_{sender} \leftarrow view\ sender \\
id_{\text{LAD}} \quad\ \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{LAD}) \\
sure\ (id_{sender} \equiv id_{\text{LAD}})
\end{array}
$$

Fail if the CDP is not currently overcollateralized.

$$\texttt{gaze}\ id_{\text{URN}} \gg\!= sure \circ (\equiv \texttt{Pride})$$

Preliminarily reduce the CDP debt.

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

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

$$
\begin{array}{l}
\texttt{pull}\ id_{\text{DAI}}\ id_{\text{LAD}}\ \text{WAD}_{\text{DAI}} \\
burn\ id_{\text{DAI}}\ \text{WAD}_{\text{DAI}}
\end{array}
$$

5.3.6 **`draw`** — issue dai

$$\texttt{draw}\ id_{\text{URN}}\ \text{WAD}_{\text{DAI}} =$$

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

Fail if sender is not the CDP owner.

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

Tentatively record DAI debt.

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

Fail if CDP with new debt is not overcollateralized.

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

Mint DAI and send it to the CDP owner.

$$\begin{aligned}
&\texttt{mint}\ id_{\text{DAI}}\ \text{WAD}_{\text{DAI}}\\
&\texttt{push}\ id_{\text{DAI}}\ id_{\text{LAD}}\ \text{WAD}_{\text{DAI}}
\end{aligned}$$

5.3.7 **free** — release collateral

$\texttt{free}\ id_{\text{URN}}\ \text{WAD}_{\text{GEM}} =$

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

Fail if sender is not the CDP owner.

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

Tentatively record the decreased collateral.

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

Fail if collateral decrease results in undercollateralization.

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

Send the collateral to the CDP owner.

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

5.4 Acts performed by price feeds

5.4.1 **mark** — update dai market price

$\texttt{mark}\ id_{\text{JAR}}\ \text{TAG}_1\ \text{ZZZ}_1 =$

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

$$\begin{array}{l} \quad jarAt\ id_{\text{JAR}} \circ \text{TAG} := \text{TAG}_1 \\ \quad jarAt\ id_{\text{JAR}} \circ \text{ZZZ} \ := \text{ZZZ}_1 \end{array}$$

$$\begin{array}{l} \texttt{mark} :: \\ \quad (\text{IsAct}, \text{Fails}\ m, \text{Logs}\ m, \\ \quad\ \ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ (\text{Id Lad}), \\ \quad\ \ \text{Writes}\ w\ m, \text{HasVat}\ w\ \text{VAT}_w, \\ \qquad\qquad\qquad \text{HasJars}\ \text{VAT}_w\ (\text{Map}\ (\text{Id Jar})\ \text{JAR}_w), \\ \qquad\qquad\qquad\quad \text{HasTag}\ \text{JAR}_w\ \text{WAD}, \\ \qquad\qquad\qquad\quad \text{HasZzz}\ \text{JAR}_w\ \text{NAT}) \\ \quad \Rightarrow \text{Id Jar} \to \text{WAD} \to \text{NAT} \to m\ () \end{array}$$

5.4.2 `tell` — update collateral market price

$$\begin{array}{l} \texttt{tell}\ x = \\ \quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\ \qquad \text{VAT} \circ \text{FIX} := x \end{array}$$

$$\begin{array}{l} \texttt{tell} :: \\ \quad (\text{IsAct}, \text{Fails}\ m, \text{Logs}\ m, \\ \quad\ \ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ (\text{Id Lad}), \\ \quad\ \ \text{Writes}\ w\ m, \text{HasVat}\ w\ \text{VAT}_w, \\ \qquad\qquad\qquad \text{HasFix}\ \text{VAT}_w\ \text{WAD}) \\ \quad \Rightarrow \text{WAD} \to m\ () \end{array}$$

5.5 Acts performed by keepers

5.5.1 **prod** — adjust target price

$$
\begin{aligned}
&\texttt{prod} = \texttt{note}\ \$\ \textbf{do}\\
&\quad \text{ERA}_0 \leftarrow \textit{view}\ \text{ERA}\\
&\quad \text{TAU}_0 \leftarrow \textit{view}\ (\text{VAT} \circ \text{TAU})\\
&\quad \text{FIX}_0 \leftarrow \textit{view}\ (\text{VAT} \circ \text{FIX})\\
&\quad \text{PAR}_0 \leftarrow \textit{view}\ (\text{VAT} \circ \text{PAR})\\
&\quad \text{HOW}_0 \leftarrow \textit{view}\ (\text{VAT} \circ \text{HOW})\\
&\quad \text{WAY}_0 \leftarrow \textit{view}\ (\text{VAT} \circ \text{WAY})\\
&\quad \textbf{let}\\
&\qquad \text{Time difference in seconds}\\
&\qquad \textit{fan} = \text{ERA}_0 - \text{TAU}_0\\
&\qquad \text{Current deflation rate applied to target price}\\
&\qquad \text{PAR}_1 = \text{PAR}_0 * \textit{cast}\ (\text{WAY}_0 \uparrow\uparrow \textit{fan})\\
&\qquad \text{Sensitivity applied over time difference}\\
&\qquad \textit{wag} = \text{HOW}_0 * \textit{fromIntegral}\ \textit{fan}\\
&\qquad \text{Deflation rate scaled up or down}\\
&\qquad \text{WAY}_1 = \textit{inj}\ (\textit{prj}\ \text{WAY}_0 +\\
&\qquad\qquad\qquad \textbf{if}\ \text{FIX}_0 < \text{PAR}_0\ \textbf{then}\ \textit{wag}\ \textbf{else} - \textit{wag})\\
&\quad \text{VAT} \circ \text{PAR} := \text{PAR}_1\\
&\quad \text{VAT} \circ \text{WAY} := \text{WAY}_1\\
&\quad \text{VAT} \circ \text{TAU} := \text{ERA}_0\\
&\quad \textbf{where}\\
&\qquad \text{Convert between multiplicative and additive form}\\
&\qquad \textit{prj}\ x = \textbf{if}\ x \geqslant 1\ \textbf{then}\ x - 1\ \textbf{else}\ 1 - 1 / x\\
&\qquad \textit{inj}\ x = \textbf{if}\ x \geqslant 0\ \textbf{then}\ x + 1\ \textbf{else}\ 1 / (1 - x)
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{prod} ::\\
&\quad (\text{IsAct}, \text{Logs}\ m,\\
&\quad\ \text{Reads}\ r\ m,\\
&\qquad \text{HasSender}\ r\ (\text{Id}\ \text{LAD}),\\
&\qquad \text{HasEra}\ r\ \text{NAT},\\
&\qquad \text{HasVat}\ r\ \text{VAT}_r,\ (\text{HasPar}\ \text{VAT}_r\ \text{WAD},\\
&\qquad\qquad\qquad\qquad \text{HasTau}\ \text{VAT}_r\ \text{NAT},\\
&\qquad\qquad\qquad\qquad \text{HasHow}\ \text{VAT}_r\ \text{RAY},\\
&\qquad\qquad\qquad\qquad \text{HasWay}\ \text{VAT}_r\ \text{RAY},
\end{aligned}
$$

$$\begin{aligned}
&\qquad\qquad\qquad\quad \text{HasFix } \textsc{vat}_r\ \textsc{wad}),\\
&\text{Writes } w\ m,\\
&\quad \text{HasVat } w\ \textsc{vat}_w, (\text{HasPar } \textsc{vat}_w\ \textsc{wad},\\
&\qquad\qquad\qquad\quad \text{HasWay } \textsc{vat}_w\ \textsc{ray},\\
&\qquad\qquad\qquad\quad \text{HasTau } \textsc{vat}_w\ \textsc{nat}),\\
&\text{Integral } \textsc{nat},\\
&\text{Ord } \textsc{wad}, \text{Fractional } \textsc{wad},\\
&\text{Fractional } \textsc{ray}, \text{Real } \textsc{ray})\\
&\Rightarrow m\ ()
\end{aligned}$$

5.5.2 **poke** — update CDP debt

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

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

Read previous stability fee accumulator.

$$\begin{aligned}
&id_{\text{ILK}} \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{ILK})\\
&phi0 \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{PHI})\\
&ice \leftarrow need\ (ilkAt\ id_{\text{ILK}} \circ \text{BAG} \circ ix\ phi0)
\end{aligned}$$

Update the stability fee accumulator.

$$\begin{aligned}
&\text{CON}_0 \leftarrow need\ (urnAt\ id_{\text{URN}} \circ \text{CON})\\
&dew \leftarrow \texttt{drip}\ id_{\text{ILK}}
\end{aligned}$$

Apply new stability fee to CDP debt.

$$urnAt\ id_{\text{URN}} \circ \text{CON} *= cast\ (dew\ /\ ice)$$

Record the poke time.

$$\begin{aligned}
&\text{ERA}_0 \leftarrow view\ \text{ERA}\\
&urnAt\ id_{\text{URN}} \circ \text{PHI} := \text{ERA}_0
\end{aligned}$$

5.5.3 **bite** — trigger CDP liquidation

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

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

Fail if urn is not undercollateralized.

$$\texttt{gaze}\ id_{\text{URN}} \gg\!= sure \circ (\equiv \texttt{Panic})$$

Record the sender as the liquidation initiator.

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

Read current debt.

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

Read liquidation penalty ratio.

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

Apply liquidation penalty to debt.

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

Update debt and record it as in need of settlement.

$\mathit{urnAt}\ \mathit{id}_{\text{URN}} \circ \text{CON} := \text{CON}_1$
$\text{SIN} \mathrel{+}= \text{CON}_1$

5.6 Acts performed by settler

5.6.1 `grab` — promise to liquidate CDP

$\texttt{grab}\ \mathit{id}_{\text{URN}} =$

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

Fail if CDP liquidation is not initiated.

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

Record the sender as the CDP's settler.

$\mathit{id}_{\text{VOW}} \leftarrow \mathit{view\ sender}$
$\mathit{urnAt}\ \mathit{id}_{\text{URN}} \circ \text{VOW} := \mathit{id}_{\text{VOW}}$

Nullify the CDP's debt and collateral.

$\text{PRO}_0 \leftarrow \mathit{need}\ (\mathit{urnAt}\ \mathit{id}_{\text{URN}} \circ \text{PRO})$
$\mathit{urnAt}\ \mathit{id}_{\text{URN}} \circ \text{CON} := 0$
$\mathit{urnAt}\ \mathit{id}_{\text{URN}} \circ \text{PRO} := 0$

Send the collateral to the settler for auctioning.

$\mathit{id}_{\text{ILK}} \leftarrow \mathit{need}\ (\mathit{urnAt}\ \mathit{id}_{\text{URN}} \circ \text{ILK})$
$\mathit{id}_{\text{JAR}} \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\text{ILK}} \circ \text{JAR})$
$\texttt{push}\ \mathit{id}_{\text{JAR}}\ \mathit{id}_{\text{VOW}}\ \text{PRO}_0$

5.6.2 `heal` — process bad debt

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

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

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

5.6.3 `loot` — process stability fee revenue

$$
\begin{array}{l}
\texttt{loot}\ \text{WAD}_{\text{DAI}} = \\
\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
\qquad \text{VAT} \circ \text{PIE} \mathrel{-}= \text{WAD}_{\text{DAI}}
\end{array}
$$

5.7 Acts performed by tests

5.7.1 `warp` — travel in time

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

$$
\begin{array}{l}
\texttt{warp} :: \\
\quad (\text{IsAct}, \text{Fails}\ m, \text{Logs}\ m, \\
\quad\ \ \text{Reads}\ r\ m,\ \ \text{HasSender}\ r\ (\text{Id}\ \text{LAD}), \\
\quad\ \ \text{Writes}\ w\ m, \text{HasEra}\ w\ \text{NAT}, \\
\qquad\qquad\qquad \text{Num}\ \text{NAT}) \\
\quad\ \Rightarrow \text{NAT} \to m\ ()
\end{array}
$$

5.8 Acts performed by tokens

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

$$
\begin{array}{l}
\texttt{pull} :: \\
\quad (\text{Fails}\ m, \\
\quad\ \ \text{Reads}\ r\ m, \\
\qquad \text{HasVat}\ r\ \text{VAT}_r,\ \ \text{HasJars}\ \text{VAT}_r\ (\text{Map}\ (\text{Id}\ \text{JAR})\ \text{JAR}_r), \\
\qquad\qquad\qquad \text{HasGem}\ \text{JAR}_r\ \text{GEM}, \\
\quad\ \ \text{Writes}\ w\ m,
\end{array}
$$

$$
\begin{aligned}
&\quad\ \mathrm{HasVat}\ w\ \textsc{vat}_w, \mathrm{HasJars}\ \textsc{vat}_w\ (\mathrm{Map}\ (\mathrm{Id}\ \textsc{Jar})\ \textsc{jar}_r))\\
&\Rightarrow \mathrm{Id}\ \textsc{Jar} \to \mathrm{Id}\ \textsc{Lad} \to \textsc{Wad} \to m\ ()
\end{aligned}
$$

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

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

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

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

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

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

5.9 System model actions

$$newLad\ id_{\text{LAD}} = lads \circ at\ id_{\text{LAD}}\ ?\!= \text{LAD}$$

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

$$\begin{aligned}&newJar\ id\ id_{\text{JAR}} =\\ &\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do}\\ &\qquad \text{VAT} \circ \text{JAR}s \circ at\ id\ ?\!= id_{\text{JAR}}\end{aligned}$$

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

5.10 Other stuff

$$\begin{aligned}&perform :: \text{Act} \to \text{Maker}\ ()\\ &perform\ x =\\ &\quad \textbf{let}\ ?act = x\ \textbf{in case}\ x\ \textbf{of}\\ &\qquad \text{NewLad}\ id \to newLad\ id\\ &\qquad \text{NewJar}\ id\ \text{JAR} \to newJar\ id\ \text{JAR}\\ &\qquad \texttt{Form}\ id\ \text{JAR} \to \texttt{form}\ id\ \text{JAR}\\ &\qquad \texttt{Mark}\ \text{JAR TAG ZZZ} \to \texttt{mark}\ \text{JAR TAG ZZZ}\\ &\qquad \texttt{Open}\ id\ \text{ILK} \to \texttt{open}\ id\ \text{ILK}\\ &\qquad \texttt{Tell}\ \text{WAD} \to \texttt{tell}\ \text{WAD}\\ &\qquad \texttt{Frob}\ \text{RAY} \to \texttt{frob}\ \text{RAY}\\ &\qquad \texttt{Prod} \to \texttt{prod}\\ &\qquad \texttt{Warp}\ t \to \texttt{warp}\ t\\ &\qquad \texttt{Give}\ \text{URN LAD} \to \texttt{give}\ \text{URN LAD}\\ &\qquad \texttt{Pull}\ \text{JAR LAD WAD} \to \texttt{pull}\ \text{JAR LAD WAD}\end{aligned}$$

```
    Lock URN WAD → lock URN WAD
transferFrom
  ::  (MonadError Error m)
  ⇒  Id LAD → Id LAD → WAD
  →  GEM → m GEM
transferFrom src dst WAD GEM =
  case GEM ^. balanceOf ∘ (at src) of
    Nothing →
      throwError AssertError
    Just balance → do
      sure (balance ⩾ WAD)
      return $ GEM
         & balanceOf ∘ ix src −~ WAD
         & balanceOf ∘ at dst %~
           (λcase
             Nothing → Just WAD
             Just x    → Just (WAD + x))
```

Chapter 6

Testing