MAKER

presents the

REFERENCE IMPLEMENTATION

of the remarkable

DAI CREDIT SYSTEM

issuing a diversely collateralized stablecoin

with last update on March 6, 2017.

Contents

1 Introduction **5**
- 1.1 Reference implementation . . . 6
- 1.2 Prerequisite Haskell knowledge . . . 7

I Implementation 9

2 Preamble **11**

3 Types **13**
- 3.1 Numeric types . . . 13
- 3.2 Identifier type . . . 14
- 3.3 Domain types . . . 14
  - Address — Ethereum accounts . . . 14
  - `Gem` — ERC20 token model . . . 15
  - `Jar` — collateral type . . . 15
  - `Ilk` — CDP type . . . 15
  - `Urn` — collateralized debt position (CDP) . . . 16
  - `Vat` — CDP engine . . . 16
  - System model . . . 17
- 3.4 Default data . . . 17

4 Act framework **21**
- 4.1 Act descriptions . . . 21
- 4.2 The Maker monad . . . 22
- 4.3 Constraints . . . 23
- 4.4 Accessor aliases . . . 23
- 4.5 Logging and asserting . . . 23
- 4.6 Modifiers . . . 23
  - `note` — logging actions . . . 23
  - `auth` — authenticating actions . . . 24

5 Acts 25
5.1 Risk assessment . . . 26
`gaze` — identify CDP risk stage . . . 26
5.2 Lending . . . 27
`open` — create CDP account . . . 27
`lock` — deposit collateral . . . 27
`free` — withdraw collateral . . . 27
`draw` — issue dai as debt . . . 28
`wipe` — repay debt and burn dai . . . 28
`give` — transfer CDP account . . . 29
`shut` — wipe, free, and delete CDP . . . 29
5.3 Frequent adjustments . . . 31
`prod` — perform revaluation and rate adjustment . . . 31
`drip` — update price of debt coin . . . 31
5.4 Governance . . . 32
`form` — create a new CDP type . . . 32
`frob` — set the sensitivity parameter . . . 32
5.5 Price feedback . . . 32
`mark` — update market price of dai . . . 32
`tell` — update market price of collateral token . . . 32
5.6 Liquidation and settlement . . . 33
`bite` — mark CDP for liquidation . . . 33
`grab` — take tokens to begin CDP liquidation . . . 33
`heal` — process bad debt . . . 33
`loot` — process stability fee revenue . . . 34
5.7 Minting, burning, and transferring . . . 34
`pull` — take tokens to vat . . . 34
`push` — send tokens from vat . . . 34
`mint` — increase supply . . . 34
`burn` — decrease supply . . . 34
5.8 Test-related manipulation . . . 34
`warp` — travel in time . . . 34
5.9 Other stuff . . . 35

6 Testing 37

A Act type signatures 39

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 declare the program's dependencies up front. The reader should probably skim this section and consult it later if unfamiliar with some type or function.

module Maker **where**

We use a typical composition of monad transformers from the `mtl` library to structure stateful actions. This becomes relevant in section 4.2 (*The Maker monad*).

import Control.Monad.State (	
MonadState,	Type class of monads with state
StateT,	Type constructor that adds state to a monad type
execStateT,	Runs a state monad with given initial state
get,	Gets the state in a **do** block
put)	Sets the state in a **do** block
import Control.Monad.Reader (	
MonadReader,	Type class of monads with "environments"
ask,	Reads the environment in a **do** block
local)	Runs a sub-computation with a modified environment
import Control.Monad.Writer (	
MonadWriter,	Type class of monads that emit logs
WriterT,	Type constructor that adds logging to a monad type
runWriterT)	Runs a writer monad
import Control.Monad.Except (	
MonadError,	Type class of monads that fail
Except,	Type constructor of failing monads
throwError,	Short-circuits the monadic computation
runExcept)	Runs a failing monad

Our numeric types use decimal fixed-point arithmetic.

import Data.Fixed (
  Fixed, Type constructor for fixed-point numbers of given precision
  HasResolution (..)) Type class for specifying precisions

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

import Control.Lens (

makeFields,	Defines lenses for record fields
view, *preview*,	Reads a lens in a **do** block
(&~),	Lets us use a **do** block with setters ◊ *Get rid of this.*
ix,	Lens for map retrieval and updating
at,	Lens for map insertion
Operators for partial state updates in **do** blocks:	
(:=),	Replace
(−=), (+=), (∗=),	Update arithmetically
(%=),	Update according to function
(?=))	Insert into map

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

import Data.Map (

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

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

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

Some less interesting imports are omitted from this document.

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

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).

Dummy types for specifying precisions

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 that combine `wad`s and `ray`s, we have to convert between the number types. Haskell does not convert numbers automatically, so when we explicitly need it, we use a *cast* function.

Convert via fractional n/m form.

$$
\begin{aligned}
&\mathit{cast} :: (\mathrm{Real}\ a, \mathrm{Fractional}\ b) \Rightarrow a \to b \\
&\mathit{cast} = \mathit{fromRational} \circ \mathit{toRational}
\end{aligned}
$$

We also define a type for non-negative integers.

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

3.2 Identifier type

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

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

$$
\begin{aligned}
&\textbf{data}\ \mathrm{Id}\ a = \mathrm{Id}\ \mathrm{String} \\
&\quad \textbf{deriving}\ (\mathrm{Show}, \mathrm{Eq}, \mathrm{Ord})
\end{aligned}
$$

We will often use mappings from IDs to the value type corresponding to that ID type, so we define an alias for such mappings.

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

3.3 Domain types

This section introduces the records stored by the Maker system. The order of presentation is by use; types further down refer to types further up, but not the other way around.

$$
\begin{aligned}
&\textbf{data}\ \mathrm{Address} = \mathrm{Address}\ \mathrm{String} \\
&\quad \textbf{deriving}\ (\mathrm{Ord}, \mathrm{Eq}, \mathrm{Show})
\end{aligned}
$$

We also have three predefined entities:

The DAI token address

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

The CDP engine address

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

The account with ultimate authority

◊ *Kludge until authority is modelled*

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

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

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

```
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,
    -- Total debt in dai
    ilkDin :: !Wad,
    -- Price of debt coin
    ilkChi :: !Ray
  } deriving (Eq, Show)
makeFields ''Ilk

data Urn = Urn {
    -- Address of biting cat
    urnCat :: !(Maybe Address),
    -- Address of liquidating vow
    urnVow :: !(Maybe Address),
    -- Issuer
    urnLad :: !Address,
    -- CDP type
    urnIlk :: !(Id Ilk),
    -- Outstanding debt in debt coins
    urnArt :: !Wad,
    -- Collateral amount in debt coins
    urnJam :: !Wad
  } deriving (Eq, Show)
makeFields ''Urn

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
    vatJars  :: !(IdMap Jar),
  CDP types
    vatIlks  :: !(IdMap Ilk),
  CDPs
    vatUrns  :: !(IdMap Urn)
  } deriving (Eq, Show)
makeFields ''Vat
```

```
data System =
  System {
    systemVat    :: Vat,
    systemEra    :: !Nat,
    systemSender :: Address
  } deriving (Eq, Show)
makeFields ''System
```

3.4 Default data

```
defaultIlk :: Id Jar → Ilk
defaultIlk id_jar = Ilk {
  ilkJar  = id_jar,
```

$$
\begin{aligned}
&\quad \mathit{ilkAxe} &&= \texttt{Ray}\ 1,\\
&\quad \mathit{ilkMat} &&= \texttt{Ray}\ 1,\\
&\quad \mathit{ilkTax} &&= \texttt{Ray}\ 1,\\
&\quad \mathit{ilkHat} &&= \texttt{Wad}\ 0,\\
&\quad \mathit{ilkLag} &&= \texttt{Nat}\ 0,\\
&\quad \mathit{ilkChi} &&= \texttt{Ray}\ 1,\\
&\quad \mathit{ilkDin} &&= \texttt{Wad}\ 0,\\
&\quad \mathit{ilkRho} &&= \texttt{Nat}\ 0\\
&\}
\end{aligned}
$$

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

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

$$
\begin{array}{l}
\quad\quad \} \\
\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{initialSystem} :: \texttt{Ray} \rightarrow \mathrm{System} \\
\mathit{initialSystem}\ \texttt{how}_0 = \mathrm{System}\ \{ \\
\quad \mathit{systemVat} \quad\ \ = \mathit{initialVat}\ \texttt{how}_0, \\
\quad \mathit{systemEra} \quad\ \ = 0, \\
\quad \mathit{systemSender} = \mathit{id}_{\mathit{god}} \\
\}
\end{array}
$$

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) Address
  | 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) Address Wad
  | Shut (Id Urn)
  | Tell Wad
  | Warp Nat
  | Wipe (Id Urn) Wad
 deriving (Eq, Show)
```

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

```
data Log = LogNote Address 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)
`aver` :: Fails m ⇒ Bool → m ()
`aver` x = *unless* x (*throwError* AssertError)
need :: (Fails m, Reads r m)
  ⇒ Getting (First a) r a → m a
need f = *preview* f ≫= λ**case**
  Nothing → *throwError* AssertError
  Just x → *return* x

4.6 Modifiers

`note` ::
  (IsAct, Logs m,
   Reads r m,

$$
\begin{aligned}
&\quad\ \ \mathrm{HasSender}\ r\ \mathrm{Address}) \\
&\Rightarrow m\ a \rightarrow m\ a
\end{aligned}
$$

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

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

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

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.

```haskell
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.

```haskell
analyze era0 par0 urn0 ilk0 jar0 =
  let
    cap = view din ilk0 * cast (view chi ilk0)
    pro = view jam urn0 * view tag jar0
    con = view art urn0 * cast (view chi ilk0) * par0
    min = con * view mat ilk0
  in if
    -- Undergoing liquidation?
    | view vow urn0 /= Nothing                   -> Dread
    -- Liquidation triggered?
    | view cat urn0 /= Nothing                   -> Grief
    -- Undercollateralized?
    | pro < min                                  -> Panic
    -- Price feed expired?
    | era0 > view zzz jar0 + view lag ilk0       -> Panic
    -- Price feed in limbo?
    | view zzz jar0 < era0                       -> Worry
    -- Debt ceiling reached?
    | cap > 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.

$$
\begin{aligned}
&\texttt{gaze}\ id_{\texttt{urn}} = \textbf{do}\\
&\quad \texttt{prod}\\
&\quad id_{\texttt{ilk}} \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{ilk})\\
&\quad \texttt{drip}\ id_{\texttt{ilk}}\\
&\quad \texttt{era}_0 \leftarrow view\ \texttt{era}\\
&\quad \texttt{par}_0 \leftarrow view\ (\texttt{vat} \circ \texttt{par})\\
&\quad \texttt{urn}_0 \leftarrow need\ (urnAt\ id_{\texttt{urn}})\\
&\quad \texttt{ilk}_0 \leftarrow need\ (ilkAt\ \ (view\ \texttt{ilk}\ \texttt{urn}_0))\\
&\quad \texttt{jar}_0 \leftarrow need\ (jarAt\ \ (view\ \texttt{jar}\ \texttt{ilk}_0))\\
&\quad return\ (analyze\ \texttt{era}_0\ \texttt{par}_0\ \texttt{urn}_0\ \texttt{ilk}_0\ \texttt{jar}_0)
\end{aligned}
$$

5.2 Lending

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

$$
\begin{aligned}
&\texttt{lock}\ id_{\texttt{urn}}\ x =\\
&\quad \texttt{note}\ \$\ \textbf{do}
\end{aligned}
$$

Ensure CDP exists; identify collateral type

$$
\begin{aligned}
&\qquad id_{\texttt{ilk}} \leftarrow need\ (urnAt\ id_{\texttt{urn}} \circ \texttt{ilk})\\
&\qquad id_{\texttt{jar}} \leftarrow need\ (ilkAt\ \ id_{\texttt{ilk}} \circ \texttt{jar})
\end{aligned}
$$

Record an increase in collateral

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

Take sender's tokens

$$
\begin{aligned}
&\qquad id_{\texttt{lad}} \leftarrow view\ sender\\
&\qquad \texttt{pull}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ x
\end{aligned}
$$

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

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

Fail if sender is not the CDP owner.

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

Tentatively record the decreased collateral.

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

Fail if collateral decrease results in undercollateralization.

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

Send the collateral to the CDP owner.

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

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

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

Fail if sender is not the CDP owner

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

Update price of debt coin

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

Denominate draw amount in debt coin

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

Increase debt

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

Roll back unless overcollateralized

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

Mint dai and send to the CDP owner

$$
\begin{aligned}
&\texttt{mint}\ id_{\text{DAI}}\ \texttt{wad}_{\text{DAI}}\\
&\texttt{push}\ id_{\text{DAI}}\ id_{\texttt{lad}}\ \texttt{wad}_{\text{DAI}}
\end{aligned}
$$

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

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

Fail if sender is not the CDP owner

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

Update price of debt coin

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

Denominate dai amount in debt coin

$\mathbf{let}\ \mathtt{wad}_{\mathtt{chi}} = \mathtt{wad}_{\mathrm{DAI}}\ /\ cast\ \mathtt{chi}_1$

Roll back if the CDP is not overcollateralized

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

Reduce debt

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

Take dai from CDP owner, or roll back

$\mathtt{pull}\ id_{\mathrm{DAI}}\ id_{\mathtt{lad}}\ \mathtt{wad}_{\mathrm{DAI}}$

Destroy dai

$\mathtt{burn}\ id_{\mathrm{DAI}}\ \mathtt{wad}_{\mathrm{DAI}}$

$\mathtt{give}\ id_{\mathtt{urn}}\ id_{\mathtt{lad}} =$
$\quad \mathtt{note}\ \$\ \mathbf{do}$
$\qquad x \leftarrow need\ (urnAt\ id_{\mathtt{urn}} \circ \mathtt{lad})$
$\qquad y \leftarrow view\ sender$
$\qquad \mathtt{aver}\ (x \equiv y)$
$\qquad urnAt\ id_{\mathtt{urn}} \circ \mathtt{lad} := id_{\mathtt{lad}}$

$\mathtt{shut}\ id_{\mathtt{urn}} =$
$\quad \mathtt{note}\ \$\ \mathbf{do}$

Update price of debt coin

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

Attempt to repay all the CDP's outstanding dai

$art0 \leftarrow need\ (urnAt\ id_{\mathtt{urn}} \circ \mathtt{art})$
$\mathtt{wipe}\ id_{\mathtt{urn}}\ (art0 * cast\ \mathtt{chi}_1)$

Reclaim all the collateral

$jam0 \leftarrow need\ (urnAt\ id_{\mathtt{urn}} \circ \mathtt{jam})$
$\mathtt{free}\ id_{\mathtt{urn}}\ jam0$

Nullify the CDP

$\texttt{vat} \circ \texttt{urn}s \circ at\ id_{\texttt{urn}} := \text{Nothing}$

5.3 Frequent adjustments

$$
\begin{array}{l}
\texttt{prod} = \texttt{note}\ \$\ \mathbf{do} \\
\quad \texttt{era}_0 \leftarrow \mathit{view}\ \texttt{era} \\
\quad \texttt{tau}_0 \leftarrow \mathit{view}\ (\texttt{vat} \circ \texttt{tau}) \\
\quad \texttt{fix}_0 \leftarrow \mathit{view}\ (\texttt{vat} \circ \texttt{fix}) \\
\quad \texttt{par}_0 \leftarrow \mathit{view}\ (\texttt{vat} \circ \texttt{par}) \\
\quad \texttt{how}_0 \leftarrow \mathit{view}\ (\texttt{vat} \circ \texttt{how}) \\
\quad \texttt{way}_0 \leftarrow \mathit{view}\ (\texttt{vat} \circ \texttt{way}) \\
\quad \mathbf{let} \\
\qquad \textsf{Time difference in seconds} \\
\qquad \mathit{fan}\ = \texttt{era}_0 - \texttt{tau}_0 \\
\qquad \textsf{Current deflation rate applied to target price} \\
\qquad \texttt{par}_1 = \texttt{par}_0 * \mathit{cast}\ (\texttt{way}_0 \uparrow\uparrow \mathit{fan}) \\
\qquad \textsf{Sensitivity parameter applied over time} \\
\qquad \mathit{wag} = \texttt{how}_0 * \mathit{fromIntegral}\ \mathit{fan} \\
\qquad \textsf{Deflation rate scaled up or down} \\
\qquad \texttt{way}_1 = \mathit{inj}\ (\mathit{prj}\ \texttt{way}_0 + \\
\qquad\qquad\qquad \mathbf{if}\ \texttt{fix}_0 < \texttt{par}_0\ \mathbf{then}\ \mathit{wag}\ \mathbf{else} - \mathit{wag}) \\
\quad \texttt{vat} \circ \texttt{par} := \texttt{par}_1 \\
\quad \texttt{vat} \circ \texttt{way} := \texttt{way}_1 \\
\quad \texttt{vat} \circ \texttt{tau} := \texttt{era}_0 \\
\quad \mathbf{where} \\
\qquad \textsf{Convert between multiplicative and additive form} \\
\qquad \mathit{prj}\ x \quad = \mathbf{if}\ x \geqslant 1\ \mathbf{then}\ x - 1\ \mathbf{else}\ 1 - 1 / x \\
\qquad \mathit{inj}\ x \quad = \mathbf{if}\ x \geqslant 0\ \mathbf{then}\ x + 1\ \mathbf{else}\ 1 / (1 - x)
\end{array}
$$

This internal act happens on every *poke*. It is also invoked when governance changes the `tax` of an `ilk`.

$$
\begin{array}{l}
\texttt{drip}\ \mathit{id}_{\texttt{ilk}} = \mathbf{do} \\
\quad \textsf{Current time stamp} \\
\quad \texttt{era}_0 \leftarrow \mathit{view}\ \texttt{era} \\
\quad \texttt{rho}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\texttt{ilk}} \circ \texttt{rho}) \\
\quad \textsf{Current stability fee} \\
\quad \texttt{tax}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\texttt{ilk}} \circ \texttt{tax}) \\
\quad \textsf{Current price of debt coin}
\end{array}
$$

$$
\begin{aligned}
&\texttt{chi}_0 \leftarrow \mathit{need}\ (\mathit{ilkAt}\ \mathit{id}_{\texttt{ilk}} \circ \texttt{chi})\\
&\textbf{let}\\
&\quad \mathit{age}\ \ = \texttt{era}_0 - \texttt{rho}_0\\
&\quad \texttt{chi}_1 = \texttt{chi}_0 * \texttt{tax}_0 \uparrow\uparrow \mathit{age}\\
&\mathit{ilkAt}\ \mathit{id}_{\texttt{ilk}} \circ \texttt{chi} := \texttt{chi}_1\\
&\mathit{ilkAt}\ \mathit{id}_{\texttt{ilk}} \circ \texttt{rho} := \texttt{era}_0\\
&\mathit{return}\ \texttt{chi}_1
\end{aligned}
$$

5.4 Governance

$$
\begin{aligned}
&\texttt{form}\ \mathit{id}_{\texttt{ilk}}\ \mathit{id}_{\texttt{jar}} =\\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do}\\
&\quad\quad \texttt{vat} \circ \texttt{ilk}s \circ \mathit{at}\ \mathit{id}_{\texttt{ilk}}\ ?\!= \mathit{defaultIlk}\ \mathit{id}_{\texttt{jar}}
\end{aligned}
$$

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

5.5 Price feedback

$$
\begin{aligned}
&\texttt{mark}\ \mathit{id}_{\texttt{jar}}\ \texttt{tag}_1\ \texttt{zzz}_1 =\\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do}\\
&\quad\quad \mathit{jarAt}\ \mathit{id}_{\texttt{jar}} \circ \texttt{tag} := \texttt{tag}_1\\
&\quad\quad \mathit{jarAt}\ \mathit{id}_{\texttt{jar}} \circ \texttt{zzz} := \texttt{zzz}_1
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{tell}\ x =\\
&\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do}\\
&\quad\quad \texttt{vat} \circ \texttt{fix} := x
\end{aligned}
$$

5.6 Liquidation and settlement

```
bite id_urn =
  note $ do
    Fail if urn is not undercollateralized
    gaze id_urn ≫= aver ∘ (≡ Panic)
    Record the sender as the requester of liquidation
    id_cat               ← view sender
    urnAt id_urn ∘ cat := id_cat
    Read current debt
    art0 ← need (urnAt id_urn ∘ art)
    Update price of debt coin
    id_ilk ← need (urnAt id_urn ∘ ilk)
    chi_1 ← drip id_ilk
    Read liquidation penalty ratio
    id_ilk ← need (urnAt id_urn ∘ ilk)
    axe_0 ← need (ilkAt id_ilk ∘ axe)
    Apply liquidation penalty to debt
    let art1 = art0 ∗ axe_0
    Update debt and record it as in need of settlement
    urnAt id_urn ∘ art := art1
    sin                += art1 ∗ chi_1
```

```
grab id_urn =
  auth ∘ note $ do
    Fail if CDP is not marked for liquidation
    gaze id_urn ≫= aver ∘ (≡ Grief)
    Record the sender as the CDP's settler
    id_vow ← view sender
    urnAt id_urn ∘ vow := id_vow
    Clear the CDP's requester of liquidation
    urnAt id_urn ∘ cat := Nothing
```

```
heal wad_DAI =
```

$$
\begin{array}{l}
\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
\quad\quad \texttt{vat} \circ \texttt{sin} \mathrel{-}= \texttt{wad}_{\text{DAI}}
\end{array}
$$

$$
\begin{array}{l}
\texttt{loot}\ \texttt{wad}_{\text{DAI}} = \\
\quad \texttt{auth} \circ \texttt{note}\ \$\ \textbf{do} \\
\quad\quad \texttt{vat} \circ \texttt{pie} \mathrel{-}= \texttt{wad}_{\text{DAI}}
\end{array}
$$

5.7 Minting, burning, and transferring

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

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

$$
\begin{array}{ll}
\multicolumn{2}{l}{\texttt{mint}\ id_{\texttt{jar}}\ \texttt{wad}_0 = \textbf{do}} \\
\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ totalSupply & \mathrel{+}= \texttt{wad}_0 \\
\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ balanceOf \circ ix\ id_{\texttt{vat}} & \mathrel{+}= \texttt{wad}_0
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\texttt{burn}\ id_{\texttt{jar}}\ \texttt{wad}_0 = \textbf{do}} \\
\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ totalSupply & \mathrel{-}= \texttt{wad}_0 \\
\quad jarAt\ id_{\texttt{jar}} \circ \texttt{gem} \circ balanceOf \circ ix\ id_{\texttt{vat}} & \mathrel{-}= \texttt{wad}_0
\end{array}
$$

5.8 Test-related manipulation

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

5.9 Other stuff

```
perform :: Act → Maker ()
perform x =
  let ?act = x in case x of
    Form id jar       → form id jar
    Mark jar tag zzz  → mark jar tag zzz
    Open id ilk       → open id ilk
    Tell wad          → tell wad
    Frob ray          → frob ray
    Prod              → prod
    Warp t            → warp t
    Give urn lad      → give urn lad
    Pull jar lad wad  → pull jar lad wad
    Lock urn wad      → lock urn wad

transferFrom
  ::  (MonadError Error m)
  ⇒  Address → Address → Wad
  →  Gem → m Gem

transferFrom src dst wad gem =
  case view (balanceOf ∘ at src) gem of
    Nothing →
      throwError AssertError
    Just balance → do
      aver (balance ⩾ wad)
      return $ gem &~ do
        balanceOf ∘ ix src −= wad
        balanceOf ∘ at dst %=
          (λcase
            Nothing → Just wad
            Just x  → Just (wad + x))
```

Chapter 6

Testing

Appendix A

Act type signatures

$$
\begin{array}{l}
\textbf{type}\ \text{Numbers}\ \texttt{wad}\ \texttt{ray}\ \texttt{nat} = \\
\quad (\texttt{wad}{\sim}\texttt{Wad}, \texttt{ray}{\sim}\texttt{Ray}, \texttt{nat}{\sim}\texttt{Nat})
\end{array}
$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\text{HasTotalSupply } \mathit{gem\_r}\ \mathtt{Wad},\\
\qquad\qquad\qquad\qquad\qquad\text{HasBalanceOf } \mathit{gem\_r}\ (\text{Map Address } \mathtt{Wad}))\\
\quad \Rightarrow \text{Id } \mathtt{Jar} \to \mathtt{Wad} \to m\ ()
\end{array}
$$

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

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