MAKER

presents the

REFERENCE IMPLEMENTATION

also known as the

PURPLE PAPER

of the remarkable

DAI CREDIT SYSTEM

issuing a diversely collateralized stablecoin

◈

elucidated by

Daniel Brockman
Mikael Brockman
Nikolai Mushegian
Rain Clever

with last update on March 11, 2017.

Contents

1 Introduction **6**

1.1 Reference implementation . . . 7

1.2 Limitations . . . 8

I Implementation 9

2 Types **11**

2.1 Numeric types . . . 11

2.2 Identifiers and addresses . . . 12

2.3 `Gem` — token model . . . 12

2.4 `Jar` — collateral vaults . . . 13

`gem` — collateral token . . . 13

`tag` — market price of token . . . 13

`zzz` — expiration time of token price feed . . . 13

2.5 `Ilk` — CDP type . . . 13

`jar` — collateral token vault . . . 13

`mat` — liquidation ratio . . . 13

`axe` — liquidation penalty ratio . . . 13

`hat` — debt ceiling . . . 13

`tax` — stability fee . . . 13

`lag` — price feed limbo duration . . . 13

`rho` — time of debt unit adjustment . . . 13

`din` — total outstanding dai . . . 13

`chi` — dai value of debt unit . . . 13

2.6 `Urn` — collateralized debt position (CDP) . . . 13

`cat` — address of liquidation initiator . . . 13

`vow` — address of liquidation contract . . . 13

`lad` — CDP owner . . . 13

`ilk` — CDP type . . . 13

art — debt denominated in debt unit . . . 13
jam — collateral denominated in debt unit . . . 13
2.7 Vat — CDP engine . . . 13
fix — market price of DAI denominated in SDR . . . 13
par — target price of DAI denominated in SDR . . . 13
how — sensitivity parameter . . . 13
way — rate of target price change . . . 13
tau — time of latest target update . . . 13
joy — unprocessed stability fee revenue . . . 13
sin — bad debt from liquidated CDPs . . . 13
2.8 System model . . . 14
era — current time . . . 14
2.9 Default data . . . 15

3 Acts **17**
3.1 Assessment . . . 18
gaze — identify CDP risk stage . . . 18
3.2 Lending . . . 20
open — create CDP . . . 20
give — transfer CDP account . . . 20
lock — deposit collateral . . . 20
free — withdraw collateral . . . 20
draw — issue dai as debt . . . 21
wipe — repay debt and burn dai . . . 21
shut — wipe, free, and delete CDP . . . 22
3.3 Adjustment . . . 23
prod — adjust target price and target rate . . . 23
drip — update debt unit and unprocessed fee revenue . . . 24
3.4 Feedback . . . 24
mark — update market price of dai . . . 24
tell — update market price of collateral token . . . 24
3.5 Liquidation . . . 24
bite — mark for liquidation . . . 24
grab — take tokens for liquidation . . . 25
plop — finish liquidation returning profit . . . 25
heal — process bad debt . . . 26
love — process stability fee revenue . . . 26
3.6 Governance . . . 26
form — create a new CDP type . . . 26
frob — set the sensitivity parameter . . . 26
chop — set liquidation penalty . . . 26

`cork` — set debt ceiling . . . 26
`calm` — set limbo duration . . . 26
`cuff` — set liquidation ratio . . . 27
`crop` — set stability fee . . . 27
3.7 Treasury . . . 27
`pull` — transfer tokens to collateral vault . . . 27
`push` — transfer tokens from collateral vault . . . 27
`mint` — create tokens . . . 27
`burn` — destroy tokens . . . 27
3.8 Manipulation . . . 28
`warp` — travel through time . . . 28
`mine` — create toy token type . . . 28
`hand` — give toy tokens to account . . . 28
`sire` — register a new toy account . . . 28
3.9 Other stuff . . . 28

4 Act framework **30**
4.1 Act descriptions . . . 30
4.2 The Maker monad . . . 31
4.3 Asserting . . . 32
4.4 Modifiers . . . 32
`auth` — authenticating actions . . . 32

5 Testing **33**

A Prelude **35**

B Rounding fixed point numbers **38**

List of Tables

2.1 CDP record 14
3.1 CDP acts in the five stages of risk 19

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 credit incentives in order to keep its 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. Thus all outstanding dai represents some CDP owner's claim on their collateral. The debt and collateral amounts are recorded in a *collateralized debt position*, or CDP.

Off-chain *price feeds* give Maker knowledge of the market values of dai and the various tokens used as collateral, enabling the system to assess credit risk. If the value of a CDP's collateral drops below a certain multiple of its debt, a decentralized liquidation auction is triggered to sell the collateral for dai to be burned thus settling the debt.

The system issues a separate token with symbol MKR, which behaves like a "share" in Maker itself. When a collateral auction fails to recover the full debt value, the MKR token is diluted by way of a *reverse auction*. The value of MKR, though volatile by design, is backed by the revenue from a *stability fee* imposed on all dai loans and used to buy MKR for burning.

This document is an executable technical specification of the of the Maker smart contracts. It is a draft; be aware that the contents will certainly change before the public launch of the dai.

[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.

1.1 Reference implementation

The version of this system that will be deployed on the blockchain is written in Solidity, which is a workable smart contract implementation language. This reference implementation is a 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 powerful testing tools such as QuickCheck and SmallCheck for comprehensively verifying key properties as a middle ground between unit testing and formal verification.

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

4. **Typing**. 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 Limitations

This implementation has a simplified model of Maker's governance authorization. Instead of the "access control list" approach of the `DSGuard` component, we give full authority to one single address. A future iteration will include the full authorization model.

We also do not currently model the EVM's 256 bit word size, but allow all quantities to grow arbitrarily large. This will also be modelled in a future iteration.

Finally, our model of ERC20 tokens is simplified, and for example does not include the concept of "allowances."

Part I

Implementation

Preamble

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

```
module Maker where
import Prelude ()        Import nothing from Prelude
import Maker.Prelude     Import everything from Maker Prelude
```

We also import our definition of decimal fixed point numbers, listed in Appendix B.

```
import Maker.Decimal
```

Chapter 2

Types

We now define the data types used by Maker: numeric types, identifiers, on-chain records, and test model data.

2.1 Numeric types

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

Define the distinct type for currency quantities

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

Define the distinct type for rates and ratios

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

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

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

Haskell number types are not automatically converted, so we convert explicitly with a *cast* function.

Convert via fractional n/m form.

$$cast :: (\text{Real } a, \text{Fractional } b) \Rightarrow a \to b$$
$$cast = fromRational \;.\; toRational$$

2.2 Identifiers and addresses

There are several kinds of identifiers used in the system, and we use types to distinguish them. The type parameter a creates distinct types; e.g., Id Foo and Id Bar are incompatible.

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

We define another type for representing Ethereum account addresses.

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

We also have two predefined entity identifiers.

The DAI token vault address

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

A test account with ultimate authority

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

2.3 `Gem` — token model

In this model, all tokens behave in the same simple way.[1] We omit the ERC20 concept of "allowances."

Tokens can be held by CDP owners, by collateral vaults, or by the test driver. We model this distinction with a data type.

$$\textbf{data}\ \text{Holder} = \text{InAccount Address} \mid \text{InVault}\ (\text{Id}\ \texttt{Jar}) \mid \text{InToy}$$
$$\quad \textbf{deriving}\ (\text{Eq}, \text{Show}, \text{Ord})$$

[1] In the real world, token semantics can differ, despite nominally following the ERC20 interface. Maker governance therefore involves due diligence on collateral token contracts.

We now define a `Gem` as simply a map keeping track of the currency amount held by each holder.

```
data Gem = Gem {• balanceOf :: Map Holder Wad}
  deriving (Eq, Show)
```

2.4 Jar — collateral vaults

```
data Jar = Jar {
  • gem :: Gem,   Collateral token
  • tag :: Wad,   Market price
  • zzz :: Sec    Price expiration
  } deriving (Eq, Show)
```

2.5 Ilk — CDP type

```
data Ilk = Ilk {
  • jar :: Id Jar,   Collateral vault
  • mat :: Ray,      Liquidation ratio
  • axe :: Ray,      Liquidation penalty
  • hat :: Wad,      Debt ceiling
  • tax :: Ray,      Stability fee
  • lag :: Sec,      Price feed limbo duration
  • rho :: Sec,      Time of latest debt unit adjustment
  • rum :: Wad,      Total debt in debt unit
  • chi :: Ray       Dai value of debt unit
  } deriving (Eq, Show)
```

2.6 Urn — collateralized debt position (CDP)

2.7 Vat — CDP engine

```
data Vat = Vat {
```

Table 2.1: CDP record

data `Urn` = `Urn` {

• `cat` :: Maybe Address,	Address of liquidation initiator
• `vow` :: Maybe Address,	Address of liquidation contract
• `lad` :: Address,	Issuer
• `ilk` :: Id `Ilk`,	CDP type
• `art` :: `Wad`,	Outstanding debt in debt unit
• `jam` :: `Wad`	Collateral amount in debt unit

} **deriving** (Eq, Show)

• `fix` :: `Wad`,	Market price
• `how` :: `Ray`,	Sensitivity
• `par` :: `Wad`,	Target price
• `way` :: `Ray`,	Target rate
• `tau` :: `Sec`,	Last prodded
• `joy` :: `Wad`,	Unprocessed stability fees
• `sin` :: `Wad`,	Bad debt from liquidated CDPs
• `jars` :: Map (Id `Jar`) `Jar`,	Collateral tokens
• `ilks` :: Map (Id `Ilk`) `Ilk`,	CDP types
• `urns` :: Map (Id `Urn`) `Urn`	CDPs

} **deriving** (Eq, Show)

2.8 System model

data System = System {

• `vat` :: `Vat`,	Root Maker entity
• `era` :: `Sec`,	Current time stamp
• `sender` :: Address,	Sender of current act
• `accounts` :: [Address]	For test suites

} **deriving** (Eq, Show)

Lens fields

makeLenses '' Gem
makeLenses '' Jar
makeLenses '' Ilk
makeLenses '' Urn
makeLenses '' Vat
makeLenses '' System

2.9 Default data

defaultIlk :: Id Jar → Ilk
defaultIlk $id_{\texttt{jar}}$ = Ilk {
- jar = $id_{\texttt{jar}}$,
- axe = Ray 1,
- mat = Ray 1,
- tax = Ray 1,
- hat = Wad 0,
- lag = Sec 0,
- chi = Ray 1,
- rum = Wad 0,
- rho = Sec 0

}

emptyUrn :: Id Ilk → Address → Urn
emptyUrn $id_{\texttt{ilk}}$ $id_{\texttt{lad}}$ = Urn {
- vow = Nothing,
- cat = Nothing,
- lad = $id_{\texttt{lad}}$,
- ilk = $id_{\texttt{ilk}}$,
- art = Wad 0,
- jam = Wad 0

}

initialVat :: Ray → Vat
initialVat $\texttt{how}_0$ = Vat {

- $\mathtt{tau} = 0$,
- $\mathtt{fix} = \mathtt{Wad}\ 1$,
- $\mathtt{par} = \mathtt{Wad}\ 1$,
- $\mathtt{how} = \mathtt{how}_0$,
- $\mathtt{way} = \mathtt{Ray}\ 1$,
- $\mathtt{joy} = \mathtt{Wad}\ 0$,
- $\mathtt{sin} = \mathtt{Wad}\ 0$,
- $\mathtt{ilks} = \varnothing$,
- $\mathtt{urns} = \varnothing$,
- $\mathtt{jars} =$
  $\mathit{singleton}\ \mathit{id}_{\text{DAI}}\ \mathtt{Jar}\ \{$
  - $\mathtt{gem} = \mathtt{Gem}\ \{$
    - $\mathtt{balanceOf} = \varnothing$

    $\}$,
  - $\mathtt{tag} = \mathtt{Wad}\ 0$,
  - $\mathtt{zzz} = 0$

  $\}$

$\}$

$\mathit{initialSystem} :: \mathrm{Ray} \to \mathrm{System}$
$\mathit{initialSystem}\ \mathtt{how}_0 = \mathrm{System}\ \{$

- $\mathtt{vat} = \mathit{initialVat}\ \mathtt{how}_0$,
- $\mathtt{era} = 0$,
- $\mathtt{sender} = \mathit{id}_{god}$,
- $\mathtt{accounts} = \mathit{mempty}$

$\}$

Chapter 3

Acts

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

For details on the underlying “Maker monad,” which specifies how the act definitions behave with regard to state and rollback thereof, see chapter 4.

3.1 Assessment

In order to prohibit CDP acts based on risk situation, we define five stages of risk.

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

We define the function *analyze* that determines the risk stage of a CDP.

$analyze\ \texttt{era}_0\ \texttt{par}_0\ \texttt{urn}_0\ \texttt{ilk}_0\ \texttt{jar}_0 =$

$\quad \textbf{if} \mid view\ \texttt{vow}\ \texttt{urn}_0 \not\equiv \text{Nothing}$

CDP liquidation in progress

$\quad\quad \to \texttt{Dread}$

$\quad \mid view\ \texttt{cat}\ \texttt{urn}_0 \not\equiv \text{Nothing}$

CDP liquidation triggered

$\quad\quad \to \texttt{Grief}$

$\quad \mid \texttt{pro} < \texttt{min}$

CDP's collateralization below liquidation ratio

$\quad\quad \to \texttt{Panic}$

$\quad \mid view\ \texttt{zzz}\ \texttt{jar}_0 + view\ \texttt{lag}\ \texttt{ilk}_0 < \texttt{era}_0$

CDP type's price limbo exceeded limit

$\quad\quad \to \texttt{Panic}$

$\quad \mid view\ \texttt{zzz}\ \texttt{jar}_0 < \texttt{era}_0$

CDP type's price feed in limbo

$\quad\quad \to \texttt{Worry}$

$\quad \mid \texttt{cap} > view\ \texttt{hat}\ \texttt{ilk}_0$

CDP type's debt ceiling exceeded

$\quad\quad \to \texttt{Anger}$

$\quad \mid otherwise$

No problems

$\quad\quad \to \texttt{Pride}$

where

CDP's collateral value in SDR:

$\texttt{pro} = view\ \texttt{jam}\ \texttt{urn}_0 * view\ \texttt{tag}\ \texttt{jar}_0$

CDP type's total debt in SDR:

$\texttt{cap} = view\ \texttt{rum}\ \texttt{ilk}_0 * cast\ (view\ \texttt{chi}\ \texttt{ilk}_0)$

CDP's debt in SDR:

$\texttt{con} = view\ \texttt{art}\ \texttt{urn}_0 * cast\ (view\ \texttt{chi}\ \texttt{ilk}_0) * \texttt{par}_0$

Required collateral as per liquidation ratio:

$\texttt{min} = \texttt{con} * cast\ (view\ \texttt{mat}\ \texttt{ilk}_0)$

Table 3.1: CDP acts in the five stages of risk

	give	shut	lock	wipe	free	draw	bite	grab	plop
Pride	✌	☜	☜	☜	☝	☝	—	—	—
Anger	✌	☜	☜	☜	☝	—	—	—	—
Worry	✌	☜	☜	☜	—	—	—	—	—
Panic	✌	☜	☜	☜	—	—	✋	—	—
Grief	✌	—	—	—	—	—	—	✊	—
Dread	✌	—	—	—	—	—	—	—	✊
		decrease risk			increase risk		unwind risk		

✌ allowed for owner unconditionally
☜ allowed for owner if able to repay
☝ allowed for owner if collateralization maintained
✊ allowed for settler contract
✋ allowed for anyone

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

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

Adjust target price and target rate

$\quad \texttt{prod}$

Update debt unit and unprocessed fee revenue

$\quad id_{\texttt{ilk}} \leftarrow look\ (\texttt{vat}\ .\ \texttt{urns}\ .\ ix\ id_{\texttt{urn}}\ .\ \texttt{ilk})$

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

Read parameters for risk analysis

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

$\quad \texttt{par}_0 \leftarrow use\ \ (\texttt{vat}\ .\ \texttt{par})$

$\quad \texttt{urn}_0 \leftarrow look\ (\texttt{vat}\ .\ \texttt{urns}\ .\ ix\ id_{\texttt{urn}})$

$\quad \texttt{ilk}_0 \leftarrow look\ (\texttt{vat}\ .\ \texttt{ilks}\ .\ ix\ (view\ \texttt{ilk}\ \texttt{urn}_0))$

$\quad \texttt{jar}_0 \leftarrow look\ (\texttt{vat}\ .\ \texttt{jars}\ .\ ix\ (view\ \texttt{jar}\ \texttt{ilk}_0))$

Return risk stage of CDP

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

Acts on CDPs use gaze to prohibit increasing risk when already risky, and to freeze debt and collateral during liquidation; see Table 3.1.

3.2 Lending

Any user can open one or more accounts with the system using open, specifying a self-chosen account identifier and a CDP type.

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

Fail if account identifier is taken

$$\mathit{none}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ id_{\texttt{urn}})$$

Create a CDP record with the sender as owner

$$id_{\texttt{lad}} \leftarrow \mathit{use\ sender}$$
$$\mathit{initialize}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{at}\ id_{\texttt{urn}})\ (\mathit{emptyUrn}\ id_{\texttt{ilk}}\ id_{\texttt{lad}})$$

The owner of a CDP can transfer its ownership at any time using give.

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

Fail if sender is not the CDP owner

$$id_{\mathit{sender}} \leftarrow \mathit{use\ sender}$$
$$\mathit{owns}\ id_{\texttt{urn}}\ id_{\mathit{sender}}$$

Transfer ownership

$$\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ id_{\texttt{urn}}\ .\ \texttt{lad} := id_{\texttt{lad}}$$

Unless liquidation has been triggered for a CDP, its owner can use lock to deposit more collateral.

$$\texttt{lock}\ id_{\texttt{urn}}\ \texttt{wad}_{\texttt{gem}} = \textbf{do}$$

Fail if sender is not the CDP owner

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

Fail if liquidation initiated

$$\mathit{want}\ (\texttt{gaze}\ id_{\texttt{urn}})\ (\notin [\texttt{Grief}, \texttt{Dread}])$$

Identify collateral type

$$id_{\texttt{ilk}} \leftarrow \mathit{look}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ id_{\texttt{urn}}\ .\ \texttt{ilk})$$
$$id_{\texttt{jar}} \leftarrow \mathit{look}\ (\texttt{vat}\ .\ \texttt{ilks}\ .\ \mathit{ix}\ id_{\texttt{ilk}}\ .\ \texttt{jar})$$

Transfer tokens from owner to collateral vault

$$\texttt{pull}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ \texttt{wad}_{\texttt{gem}}$$

Record an increase in collateral

$$\mathit{increase}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ id_{\texttt{urn}}\ .\ \texttt{jam})\ \texttt{wad}_{\texttt{gem}}$$

When a CDP has no risk problems—except that its CDP type's debt ceiling may be exceeded—its owner can use free to withdraw some amount of collateral, as long as the withdrawal would not reduce collateralization below the liquidation ratio.

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

Fail if sender is not the CDP owner

$id_{\texttt{lad}} \leftarrow \textit{use sender}$
$\textit{owns}\ id_{\texttt{urn}}\ id_{\texttt{lad}}$

Record a decrease in collateral

$\textit{decrease}\ (\texttt{vat . urns . } \textit{ix}\ id_{\texttt{urn}}\ \texttt{. jam})\ \texttt{wad}_{\texttt{gem}}$

Roll back on any risk problem except debt ceiling excess

$\textit{want}\ (\texttt{gaze}\ id_{\texttt{urn}})\ (\in [\texttt{Pride}, \texttt{Anger}])$

Transfer tokens from collateral vault to owner

$id_{\texttt{ilk}} \leftarrow \textit{look}\ (\texttt{vat . urns . } \textit{ix}\ id_{\texttt{urn}}\ \texttt{. ilk})$
$id_{\texttt{jar}} \leftarrow \textit{look}\ (\texttt{vat . ilks . } \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. jar})$
$\texttt{push}\ id_{\texttt{jar}}\ id_{\texttt{lad}}\ \texttt{wad}_{\texttt{gem}}$

When a CDP has no risk problems, its owner can can use `draw` to take out a loan of newly minted dai, as long as the CDP type's debt ceiling is not reached and the loan would not result in undercollateralization.

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

Fail if sender is not the CDP owner

$id_{\texttt{lad}} \leftarrow \textit{use sender}$
$\textit{owns}\ id_{\texttt{urn}}\ id_{\texttt{lad}}$

Update debt unit and unprocessed fee revenue

$id_{\texttt{ilk}} \leftarrow \textit{look}\ (\texttt{vat . urns . } \textit{ix}\ id_{\texttt{urn}}\ \texttt{. ilk})$
$\texttt{chi}_1 \leftarrow \texttt{drip}\ id_{\texttt{ilk}}$

Denominate loan in debt unit

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

Increase CDP debt

$\textit{increase}\ (\texttt{vat . urns . } \textit{ix}\ id_{\texttt{urn}}\ \texttt{. art})\ \texttt{wad}_{\texttt{chi}}$

Increase total debt of CDP type

$\textit{increase}\ (\texttt{vat . ilks . } \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. rum})\ \texttt{wad}_{\texttt{chi}}$

Roll back on any risk problem

$\textit{want}\ (\texttt{gaze}\ id_{\texttt{urn}})\ (\equiv \texttt{Pride})$

Mint dai and transfer to CDP owner

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

A CDP owner who has previously loaned dai can use `wipe` to repay part of their debt as long as liquidation has not been initiated.

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

Fail if sender is not the CDP owner

$\mathit{id}_{\texttt{lad}} \leftarrow \mathit{use\ sender}$

$\mathit{owns}\ \mathit{id}_{\texttt{urn}}\ \mathit{id}_{\texttt{lad}}$

Fail if liquidation initiated

$\mathit{want}\ (\texttt{gaze}\ \mathit{id}_{\texttt{urn}})\ (\notin [\texttt{Grief}, \texttt{Dread}])$

Update debt unit and unprocessed fee revenue

$\mathit{id}_{\texttt{ilk}} \leftarrow \mathit{look}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{urn}}\ .\ \texttt{ilk})$

$\texttt{chi}_1 \leftarrow \texttt{drip}\ \mathit{id}_{\texttt{ilk}}$

Denominate dai amount in debt unit

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

Decrease CDP debt

$\mathit{decrease}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{urn}}\ .\ \texttt{art})\ \texttt{wad}_{\texttt{chi}}$

Decrease total CDP type debt

$\mathit{decrease}\ (\texttt{vat}\ .\ \texttt{ilks}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{ilk}}\ .\ \texttt{rum})\ \texttt{wad}_{\texttt{chi}}$

Transfer dai from CDP owner to dai vault

$\texttt{pull}\ \mathit{id}_{\text{DAI}}\ \mathit{id}_{\texttt{lad}}\ \texttt{wad}_{\text{DAI}}$

Destroy reclaimed dai

$\texttt{burn}\ \mathit{id}_{\text{DAI}}\ \texttt{wad}_{\text{DAI}}$

A CDP owner can use shut to close their account—repaying all debt and reclaiming all collateral—if the price feed is up to date and liquidation has not been initiated.

$\texttt{shut}\ \mathit{id}_{\texttt{urn}} = \textbf{do}$

Update debt unit and unprocessed fee revenue

$\mathit{id}_{\texttt{ilk}} \leftarrow \mathit{look}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{urn}}\ .\ \texttt{ilk})$

$\texttt{chi}_1 \leftarrow \texttt{drip}\ \mathit{id}_{\texttt{ilk}}$

Reclaim all outstanding dai

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

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

Reclaim all collateral

$\texttt{jam}_0 \leftarrow \mathit{look}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{urn}}\ .\ \texttt{jam})$

$\texttt{free}\ \mathit{id}_{\texttt{urn}}\ \texttt{jam}_0$

Nullify CDP record

$\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{at}\ \mathit{id}_{\texttt{urn}} := \text{Nothing}$

3.3 Adjustment

$$
\begin{aligned}
&\texttt{prod} = \mathbf{do} \\
&\quad \texttt{era}_0 \leftarrow \textit{use}\ \texttt{era} \\
&\quad \texttt{tau}_0 \leftarrow \textit{use}\ (\texttt{vat}\,.\,\texttt{tau}) \\
&\quad \texttt{fix}_0 \leftarrow \textit{use}\ (\texttt{vat}\,.\,\texttt{fix}) \\
&\quad \texttt{par}_0 \leftarrow \textit{use}\ (\texttt{vat}\,.\,\texttt{par}) \\
&\quad \texttt{how}_0 \leftarrow \textit{use}\ (\texttt{vat}\,.\,\texttt{how}) \\
&\quad \texttt{way}_0 \leftarrow \textit{use}\ (\texttt{vat}\,.\,\texttt{way}) \\
&\quad \mathbf{let}
\end{aligned}
$$

Time difference in seconds

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

Current target rate applied to target price

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

Sensitivity parameter applied over time

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

Target rate scaled up or down

$$
\begin{aligned}
\texttt{way}_1 = \textit{inj}\ (&\textit{prj}\ \texttt{way}_0 + \\
&\mathbf{if}\ \texttt{fix}_0 < \texttt{par}_0\ \mathbf{then}\ \textit{wag}\ \mathbf{else} - \textit{wag})
\end{aligned}
$$

$$
\begin{aligned}
&\quad \texttt{vat}\,.\,\texttt{par} := \texttt{par}_1 \\
&\quad \texttt{vat}\,.\,\texttt{way} := \texttt{way}_1 \\
&\quad \texttt{vat}\,.\,\texttt{tau} := \texttt{era}_0 \\
&\quad \mathbf{where}
\end{aligned}
$$

Convert between multiplicative and additive form

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

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

$\texttt{rho}_0 \leftarrow \textit{look}\ (\texttt{vat . ilks .}\ \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. rho})$	Time stamp of previous `drip`
$\texttt{tax}_0 \leftarrow \textit{look}\ (\texttt{vat . ilks .}\ \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. tax})$	Current stability fee
$\texttt{chi}_0 \leftarrow \textit{look}\ (\texttt{vat . ilks .}\ \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. chi})$	Current debt unit value
$\texttt{rum}_0 \leftarrow \textit{look}\ (\texttt{vat . ilks .}\ \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. rum})$	Current total debt in debt unit
$\texttt{joy}_0 \leftarrow \textit{look}\ (\texttt{vat . joy})$	Current unprocessed stability fee revenue
$\texttt{era}_0 \leftarrow \textit{use}\ \texttt{era}$	Current time stamp

let

$$\begin{aligned} \textit{age} &= \texttt{era}_0 - \texttt{rho}_0 \\ \texttt{chi}_1 &= \texttt{chi}_0 * \texttt{tax}_0 \uparrow\uparrow \textit{age} \\ \texttt{joy}_1 &= \texttt{joy}_0 + (\textit{cast}\ (\texttt{chi}_1 - \texttt{chi}_0) :: \texttt{Wad}) * \texttt{rum}_0 \end{aligned}$$

$\texttt{vat . ilks .}\ \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. chi} := \texttt{chi}_1$
$\texttt{vat . ilks .}\ \textit{ix}\ id_{\texttt{ilk}}\ \texttt{. rho} := \texttt{era}_0$
$\texttt{vat . joy} := \texttt{joy}_1$

$\textit{return}\ \texttt{chi}_1$

3.4 Feedback

$\texttt{mark}\ id_{\texttt{jar}}\ \texttt{tag}_1\ \texttt{zzz}_1 =$
$\quad \texttt{auth}\ \$\ \textbf{do}$
$\qquad \texttt{vat . jars .}\ \textit{ix}\ id_{\texttt{jar}}\ \texttt{. tag} := \texttt{tag}_1$
$\qquad \texttt{vat . jars .}\ \textit{ix}\ id_{\texttt{jar}}\ \texttt{. zzz} := \texttt{zzz}_1$

$\texttt{tell}\ \texttt{wad}_{\texttt{gem}} =$
$\quad \texttt{auth}\ \$\ \textbf{do}$
$\qquad \texttt{vat . fix} := \texttt{wad}_{\texttt{gem}}$

3.5 Liquidation

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

Fail if CDP is not in need of liquidation

$\quad \textit{want}\ (\texttt{gaze}\ id_{\texttt{urn}})\ (\equiv \texttt{Panic})$

Record the sender as the liquidation initiator

$id_{\texttt{cat}} \leftarrow \textit{use sender}$
$\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . cat} := \text{Just } id_{\texttt{cat}}$

Read current debt
$\texttt{art}_0 \leftarrow \textit{look } (\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . art})$

Update debt unit
$id_{\texttt{ilk}} \leftarrow \textit{look } (\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . ilk})$
$\texttt{chi}_1 \leftarrow \texttt{drip } id_{\texttt{ilk}}$

Read liquidation penalty ratio
$id_{\texttt{ilk}} \leftarrow \textit{look } (\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . ilk})$
$\texttt{axe}_0 \leftarrow \textit{look } (\texttt{vat . ilks . } \textit{ix } id_{\texttt{ilk}} \texttt{ . axe})$

Apply liquidation penalty to debt
$\textbf{let } \texttt{art}_1 = \texttt{art}_0 * \textit{cast } \texttt{axe}_0$

Update CDP debt
$\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . art} := \texttt{art}_1$

Record as bad debt
$\textit{increase } (\texttt{vat . sin}) \ (\texttt{art}_1 * \textit{cast } \texttt{chi}_1)$

$\texttt{grab } id_{\texttt{urn}} =$
$\quad \texttt{auth } \$ \ \textbf{do}$

Fail if CDP is not marked for liquidation
$\textit{want } (\texttt{gaze } id_{\texttt{urn}}) \ (\equiv \texttt{Grief})$

Record the sender as the CDP's settler
$id_{\texttt{vow}} \leftarrow \textit{use sender}$
$\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . vow} := \text{Just } id_{\texttt{vow}}$

Forget the CDP's requester of liquidation
$\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . cat} := \text{Nothing}$

$\texttt{plop } id_{\texttt{urn}} \ \texttt{wad}_{\text{DAI}} =$
$\quad \texttt{auth } \$ \ \textbf{do}$

Fail unless CDP is in liquidation
$\textit{want } (\texttt{gaze } id_{\texttt{urn}}) \ (\equiv \texttt{Dread})$

Forget the CDP's settler
$\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . vow} := \text{Nothing}$

Return some amount of excess auction gains
$\texttt{vat . urns . } \textit{ix } id_{\texttt{urn}} \texttt{ . jam} := \texttt{wad}_{\text{DAI}}$

$$
\begin{aligned}
&\texttt{heal}\ \texttt{wad}_{\text{DAI}} = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \textit{decrease}\ (\texttt{vat}\ .\ \texttt{sin})\ \texttt{wad}_{\text{DAI}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{love}\ \texttt{wad}_{\text{DAI}} = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \textit{decrease}\ (\texttt{vat}\ .\ \texttt{joy})\ \texttt{wad}_{\text{DAI}}
\end{aligned}
$$

3.6 Governance

$$
\begin{aligned}
&\texttt{form}\ \mathit{id}_{\texttt{ilk}}\ \mathit{id}_{\texttt{jar}} = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \textit{initialize}\ (\texttt{vat}\ .\ \texttt{ilks}\ .\ \mathit{at}\ \mathit{id}_{\texttt{ilk}}) \\
&\qquad\quad (\mathit{defaultIlk}\ \mathit{id}_{\texttt{jar}})
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{frob}\ \texttt{how}_1 = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \texttt{vat}\ .\ \texttt{how} := \texttt{how}_1
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{chop}\ \mathit{id}_{\texttt{ilk}}\ \texttt{axe}_1 = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \texttt{vat}\ .\ \texttt{ilks}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{ilk}}\ .\ \texttt{axe} := \texttt{axe}_1
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{cork}\ \mathit{id}_{\texttt{ilk}}\ \texttt{hat}_1 = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \texttt{vat}\ .\ \texttt{ilks}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{ilk}}\ .\ \texttt{hat} := \texttt{hat}_1
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{calm}\ \mathit{id}_{\texttt{ilk}}\ \texttt{lag}_1 = \\
&\quad \texttt{auth}\ \$\ \textbf{do} \\
&\qquad \texttt{vat}\ .\ \texttt{ilks}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{ilk}}\ .\ \texttt{lag} := \texttt{lag}_1
\end{aligned}
$$

```
cuff id_ilk mat₁ =
  auth $ do
    vat . ilks . ix id_ilk . mat := mat₁
```

```
crop id_ilk tax₁ =
  auth $ do
    drip id_ilk
    vat . ilks . ix id_ilk . tax := tax₁
```

3.7 Treasury

```
pull id_jar id_lad wad_gem =
  transfer id_jar wad_gem
    (InAccount id_lad)
    (InVault   id_jar)
```

```
push id_jar id_lad wad_gem =
  transfer id_jar wad_gem
    (InVault   id_jar)
    (InAccount id_lad)
```

```
mint id_jar wad₀ =
  zoom (vat . jars . ix id_jar . gem) $ do
    increase (balanceOf . ix (InVault id_jar)) wad₀
```

```
burn id_jar wad₀ =
  zoom (vat . jars . ix id_jar . gem) $ do
    decrease (balanceOf . ix (InVault id_jar)) wad₀
```

3.8 Manipulation

$\texttt{warp}\ t = \texttt{auth}\ (\textbf{do}\ \textit{increase}\ \texttt{era}\ t)$

$\texttt{mine}\ \mathit{id}_{\texttt{jar}} = \textbf{do}$
  $\textit{initialize}\ (\texttt{vat}\ .\ \texttt{jars}\ .\ \textit{at}\ \mathit{id}_{\texttt{jar}})$
    (Jar {
- gem = Gem (*singleton* InToy 1000000000000),
- tag = Wad 0,
- zzz = 0})

$\texttt{hand}\ \mathit{dst}\ \texttt{wad}_{\texttt{gem}}\ \mathit{id}_{\texttt{jar}} = \textbf{do}$
  $\textit{transfer}\ \mathit{id}_{\texttt{jar}}\ \texttt{wad}_{\texttt{gem}}$
    InToy (InAccount *dst*)

$\texttt{sire}\ \texttt{lad} = \textbf{do}\ \textit{prepend accounts}\ \texttt{lad}$

3.9 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
    Mine id          → mine id
```

$\texttt{Hand lad wad jar} \rightarrow \texttt{hand lad wad jar}$
$\texttt{Sire lad} \rightarrow \texttt{sire lad}$

$$
\begin{aligned}
&\mathit{being} :: \text{Act} \rightarrow \text{Address} \rightarrow \text{Maker}\ ()\\
&\mathit{being}\ x\ \mathit{who} = \textbf{do}\\
&\quad \mathit{old} \leftarrow \mathit{use\ sender}\\
&\quad \mathit{sender} := \mathit{who}\\
&\quad y \leftarrow \mathit{perform}\ x\\
&\quad \mathit{sender} := \mathit{old}\\
&\quad \mathit{return}\ y
\end{aligned}
$$

$\mathit{transfer}\ \mathit{id}_{\texttt{jar}}\ \texttt{wad}\ \mathit{src}\ \mathit{dst} =$

Operate in the token's balance table

$\mathit{zoom}\ (\texttt{vat}\ .\ \texttt{jars}\ .\ \mathit{ix}\ \mathit{id}_{\texttt{jar}}\ .\ \texttt{gem}\ .\ \mathit{balanceOf})\ \$\ \textbf{do}$

Fail if source balance insufficient

$\mathit{balance} \leftarrow \mathit{look}\ (\mathit{ix}\ \mathit{src})$
$\texttt{aver}\ (\mathit{balance} \geqslant \texttt{wad})$

Decrease source balance

$\mathit{decrease}\ (\mathit{ix}\ \mathit{src})\ \texttt{wad}$

Increase destination balance

$\mathit{initialize}\ (\mathit{at}\ \mathit{dst})\ 0$
$\mathit{increase}\ (\mathit{ix}\ \mathit{dst})\ \texttt{wad}$

Chapter 4

Act framework

4.1 Act descriptions

We define the Maker act vocabulary as a data type.

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

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

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

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

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 and state in a way that is still purely functional.

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


type Maker a = Maker' System a


type instance Zoomed (Maker' s) = Focusing (Except Error)
instance Zoom (Maker' s) (Maker' t) s t where
  zoom l (Maker m) = Maker (zoom l m)


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

4.3 Asserting

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

$$\begin{aligned}&\mathit{none}\ x = \mathit{preuse}\ x \ggg\!= \lambda\mathbf{case}\\&\quad \mathrm{Nothing} \to \mathit{return}\ ()\\&\quad \mathrm{Just}\ \_ \to \mathit{throwError}\ (\mathrm{AssertError}\ ?\mathit{act})\end{aligned}$$

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

$$\mathit{want}\ m\ p = m \ggg\!= (\texttt{aver}\ .\ p)$$

$$\mathit{notElem}\ x\ \mathit{xs} = \neg\ (\mathit{elem}\ x\ \mathit{xs})$$

We define *owns* $id_{\texttt{urn}}$ $id_{\texttt{lad}}$ as an assertion that the given CDP is owned by the given account.

$$\begin{aligned}&\mathit{owns}\ id_{\texttt{urn}}\ id_{\texttt{lad}} = \mathbf{do}\\&\quad \mathit{want}\ (\mathit{look}\ (\texttt{vat}\ .\ \texttt{urns}\ .\ \mathit{ix}\ id_{\texttt{urn}}\ .\ \texttt{lad}))\ (\equiv id_{\texttt{lad}})\end{aligned}$$

4.4 Modifiers

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

Chapter 5

Testing

Sketches for property stuff...

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

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

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

```
also :: Lens' s a → Lens' s b → Lens' s (a, b)
also f g = lens getter setter
  where
  getter x = (view f x, view g x)
  setter x (a, b) = set f a (set g b x)
```

```
keeps :: Parameter → Maker () → System → Bool
keeps Fix = maintains (vat . fix)
keeps Par = maintains (vat . par)
keeps Way = maintains (vat . way)
```

Thus:

```
foo sys_0 = all (λf → f sys_0)
  [changesOnly ((vat . par) `also`
                (vat . way))
     (perform Prod)]
```

Appendix A

Prelude

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

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

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

Functional operators
(.), ($),

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

Utilities
all, $\neg$, *elem*,

Constants
mempty, $\perp$, *otherwise*)

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

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

Our numeric types use decimal fixed-point arithmetic.

import Data.Fixed *as* X (	
Fixed (..),	Type constructor for numbers of given precision
HasResolution (..))	Type class for specifying precisions

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

import Control.Lens *as* X (

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

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

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

import Control.Lens.Zoom *as* X
import Control.Lens.Internal.Zoom *as* X

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

import Data.Map *as* X (

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

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

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

Some less interesting imports are omitted from this document.

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

Appendix B

Rounding fixed point numbers

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

```
module Maker.Decimal (Decimal, E18, E36, Epsilon (..)) where
import Data.Fixed
newtype HasResolution e ⇒ Decimal e = D (Fixed e)
  deriving (Ord, Eq, Real, RealFrac)
```

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

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

In the Num instance, we delegate everything except multiplication.

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

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

In the Fractional instance, we delegate everything except division.

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

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

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

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

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

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

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