Risk-Free Lending

Matthew Doty

March 17, 2025

Abstract

We construct an abstract ledger supporting the *risk-free lending* protocol. The risk-free lending protocol is a system for issuing and exchanging novel financial products we call *risk-free loans*. The system allows one party to lend money at 0% APY to another party in exchange for a good or service. On every update of the ledger, accounts have interest distributed to them. Holders of lent assets keep interest accrued by those assets. After distributing interest, the system returns a fixed fraction of each loan. These fixed fractions determine *loan periods*. Loans for longer periods have a smaller fixed fraction returned. Loans may be re-lent or used as collateral for other loans. We give a sufficient criterion to enforce that all accounts will forever be solvent. We give a protocol for maintaining this invariant when transferring or lending funds. We also show that this invariant holds after an update. Even though the system does not track counter-party obligations, we show that all credited and debited loans cancel, and the monetary supply grows at a specified interest rate.

Contents

theory *Risk-Free-Lending*
imports
Complex-Main
HOL−Cardinals.Cardinals
begin

1 Accounts

We model accounts as functions from *nat* to *real* with *finite support*.

Index *0* corresponds to an account's *cash* reserve (see §3 for details).

An index greater than *0* may be regarded as corresponding to a financial product. Such financial products are similar to *notes*. Our notes are intended to be as easy to use for exchange as cash. Positive values are debited. Negative values are credited.

We refer to our new financial products as *risk-free loans*, because they may be regarded as 0% APY loans that bear interest for the debtor. They are *risk-free* because we prove a *safety* theorem for them. Our safety theorem proves no account will "be in the red", with more credited loans than debited loans, provided an invariant is maintained. We call this invariant *strictly solvent*. See §7 for details on our safety proof.

Each risk-free loan index corresponds to a progressively shorter *loan period*. Informally, a loan period is the time it takes for 99% of a loan to be returned given a *rate function* ϱ. Rate functions are introduced in §6.

It is unnecessary to track counter-party obligations so we do not. See §4.1 and §4.2 for details.

typedef *account* = (*fin-support 0 UNIV*) :: (*nat* ⇒ *real*) *set*
⟨*proof*⟩

The type definition for *account* automatically generates two functions: *Rep-account* and *Rep-account*. *Rep-account* is a left inverse of *Abs-account*. For convenience we introduce the following shorthand notation:

notation *Rep-account* (‹π›)
notation *Abs-account* (‹ι›)

Accounts form an Abelian group. *Summing* accounts will be helpful in expressing how all credited and debited loans can cancel across a ledger. This is done in §4.1.

It is also helpful to think of an account as a transferable quantity. Transferring subtracts values under indexes from one account and adds them to another. Transfers are presented in §4.2.

instantiation *account* :: *ab-group-add*
begin

definition $0 = \iota\ (\lambda\ \text{-}\ .\ 0)$
definition $-\ \alpha = \iota\ (\lambda\ n\ .\ -\ \pi\ \alpha\ n)$
definition $\alpha_1 + \alpha_2 = \iota\ (\lambda\ n.\ \pi\ \alpha_1\ n + \pi\ \alpha_2\ n)$
definition $(\alpha_1 :: \mathit{account}) - \alpha_2 = \alpha_1 + -\ \alpha_2$

lemma *Rep-account-zero* [*simp*]: $\pi\ 0 = (\lambda\ \text{-}\ .\ 0)$
⟨*proof*⟩

lemma *Rep-account-uminus* [*simp*]:
  $\pi\ (-\ \alpha) = (\lambda\ n\ .\ -\ \pi\ \alpha\ n)$
⟨*proof*⟩

lemma *fin-support-closed-under-addition*:
  fixes $f\ g :: {'a} \Rightarrow \mathit{real}$
  assumes $f \in \mathit{fin\text{-}support}\ 0\ A$
  and $g \in \mathit{fin\text{-}support}\ 0\ A$
  shows $(\lambda\ x\ .\ f\ x + g\ x) \in \mathit{fin\text{-}support}\ 0\ A$
  ⟨*proof*⟩

lemma *Rep-account-plus* [*simp*]:
  $\pi\ (\alpha_1 + \alpha_2) = (\lambda\ n.\ \pi\ \alpha_1\ n + \pi\ \alpha_2\ n)$
  ⟨*proof*⟩

instance
⟨*proof*⟩

end

2 Strictly Solvent

An account is *strictly solvent* when, for every loan period, the sum of all the debited and credited loans for longer periods is positive. This implies that the *net asset value* for the account is positive. The net asset value is the sum of all of the credit and debit in the account. We prove *strictly-solvent* $\alpha \Longrightarrow 0 \leq$ *net-asset-value* α in §5.1.2.

definition *strictly-solvent* :: *account* ⇒ *bool* **where**
strictly-solvent $\alpha \equiv \forall\ n\ .\ 0 \leq (\sum\ i{\leq}n\ .\ \pi\ \alpha\ i)$

lemma *additive-strictly-solvent*:
assumes *strictly-solvent* α_1 **and** *strictly-solvent* α_2
shows *strictly-solvent* $(\alpha_1 + \alpha_2)$
⟨*proof*⟩

The notion of strictly solvent generalizes to a partial order, making *account* an ordered Abelian group.

instantiation *account* :: *ordered-ab-group-add*
begin

definition *less-eq-account* :: *account* ⇒ *account* ⇒ *bool* **where**
less-eq-account $\alpha_1\ \alpha_2 \equiv \forall\ n\ .\ (\sum\ i{\leq}n\ .\ \pi\ \alpha_1\ i) \leq (\sum\ i{\leq}n\ .\ \pi\ \alpha_2\ i)$

definition *less-account* :: *account* ⇒ *account* ⇒ *bool* **where**
less-account $\alpha_1\ \alpha_2 \equiv (\alpha_1 \leq \alpha_2 \land \neg\ \alpha_2 \leq \alpha_1)$

instance
⟨*proof*⟩
end

An account is strictly solvent exactly when it is *greater than or equal to 0*, according to the partial order just defined.

lemma *strictly-solvent-alt-def*: *strictly-solvent* $\alpha = (0 \leq \alpha)$
⟨*proof*⟩

3 Cash

The *cash reserve* in an account is the value under index 0.

Cash is treated with distinction. For instance it grows with interest (see §5). When we turn to balanced ledgers in §4.1, we will see that cash is the only quantity that does not cancel out.

definition *cash-reserve* :: *account* ⇒ *real* **where**
cash-reserve $\alpha = \pi\ \alpha\ 0$

If α is strictly solvent then it has non-negative cash reserves.

lemma *strictly-solvent-non-negative-cash*:
assumes *strictly-solvent* α
shows $0 \leq$ *cash-reserve* α
⟨*proof*⟩

An account consists of *just cash* when it has no other credit or debit other than under the first index.

definition *just-cash* :: *real* ⇒ *account* **where**

$just\text{-}cash\ c = \iota\ (\lambda\ n\ .\ if\ n = 0\ then\ c\ else\ 0)$

lemma *Rep-account-just-cash* [*simp*]:
$\pi\ (just\text{-}cash\ c) = (\lambda\ n\ .\ if\ n = 0\ then\ c\ else\ 0)$
⟨*proof*⟩

4 Ledgers

We model a *ledger* as a function from an index type $'a$ to an *account*. A ledger could be thought of as an *indexed set* of accounts.

type-synonym $'a\ ledger = 'a \Rightarrow account$

4.1 Balanced Ledgers

We say a ledger is *balanced* when all of the debited and credited loans cancel, and all that is left is just cash.

Conceptually, given a balanced ledger we are justified in not tracking counterparty obligations.

definition (**in** *finite*) *balanced* :: $'a\ ledger \Rightarrow real \Rightarrow bool$ **where**
$balanced\ \mathcal{L}\ c \equiv (\sum\ a \in UNIV.\ \mathcal{L}\ a) = just\text{-}cash\ c$

Provided the total cash is non-negative, a balanced ledger is a special case of a ledger which is globally strictly solvent.

lemma *balanced-strictly-solvent*:
assumes $0 \leq c$ **and** $balanced\ \mathcal{L}\ c$
shows $strictly\text{-}solvent\ (\sum\ a \in UNIV.\ \mathcal{L}\ a)$
⟨*proof*⟩

lemma (**in** *finite*) *finite-Rep-account-ledger* [*simp*]:
$\pi\ (\sum\ a \in (A :: 'a\ set).\ \mathcal{L}\ a)\ n = (\sum\ a \in A.\ \pi\ (\mathcal{L}\ a)\ n)$
⟨*proof*⟩

An alternate definition of balanced is that the *cash-reserve* for each account sums to c, and all of the other credited and debited assets cancels out.

lemma (**in** *finite*) *balanced-alt-def*:
$balanced\ \mathcal{L}\ c =$
$((\sum\ a \in UNIV.\ cash\text{-}reserve\ (\mathcal{L}\ a)) = c$
$\wedge\ (\forall\ n > 0.\ (\sum\ a \in UNIV.\ \pi\ (\mathcal{L}\ a)\ n) = 0))$
(**is** $?lhs = ?rhs$)
⟨*proof*⟩

4.2 Transfers

A *transfer amount* is the same as an *account*. It is just a function from *nat* to *real* with finite support.

type-synonym *transfer-amount* = *account*

When transferring between accounts in a ledger we make use of the Abelian group operations defined in §1.

definition *transfer* :: $'a$ *ledger* $\Rightarrow$ *transfer-amount* $\Rightarrow$ $'a \Rightarrow$ $'a \Rightarrow$ $'a$ *ledger* **where**
$$\textit{transfer}\ \mathcal{L}\ \tau\ a\ b\ x = (\textit{if}\ a = b\ \textit{then}\ \mathcal{L}\ x\ \textit{else if}\ x = a\ \textit{then}\ \mathcal{L}\ a - \tau\ \textit{else if}\ x = b\ \textit{then}\ \mathcal{L}\ b + \tau\ \textit{else}\ \mathcal{L}\ x)$$

Transferring from an account to itself is a no-op.

lemma *transfer-collapse*:
$\textit{transfer}\ \mathcal{L}\ \tau\ a\ a = \mathcal{L}$
⟨*proof*⟩

After a transfer, the sum totals of all credited and debited assets are preserved.

lemma (**in** *finite*) *sum-transfer-equiv*:
fixes $x\ y :: 'a$
shows $(\sum a \in \mathit{UNIV}.\ \mathcal{L}\ a) = (\sum a \in \mathit{UNIV}.\ \textit{transfer}\ \mathcal{L}\ \tau\ x\ y\ a)$
(**is** - = $(\sum a \in \mathit{UNIV}.\ ?\mathcal{L}'\ a)$)
⟨*proof*⟩

Since the sum totals of all credited and debited assets are preserved after transfer, a ledger is balanced if and only if it is balanced after transfer.

lemma (**in** *finite*) *balanced-transfer*:
$\textit{balanced}\ \mathcal{L}\ c = \textit{balanced}\ (\textit{transfer}\ \mathcal{L}\ \tau\ a\ b)\ c$
⟨*proof*⟩

Similarly, the sum total of a ledger is strictly solvent if and only if it is strictly solvent after transfer.

lemma (**in** *finite*) *strictly-solvent-transfer*:
fixes $x\ y :: 'a$
shows $\textit{strictly-solvent}\ (\sum a \in \mathit{UNIV}.\ \mathcal{L}\ a) =$
$\textit{strictly-solvent}\ (\sum a \in \mathit{UNIV}.\ \textit{transfer}\ \mathcal{L}\ \tau\ x\ y\ a)$
⟨*proof*⟩

4.3 The Valid Transfers Protocol

In this section we give a *protocol* for safely transferring value from one account to another.

We enforce that every transfer is *valid*. Valid transfers are intended to be intuitive. For instance one cannot transfer negative cash. Nor is it possible for an account that only has \$50 to loan out \$5,000,000.

A transfer is valid just in case the *transfer-amount* is strictly solvent and the account being credited the transfer will be strictly solvent afterwards.

definition *valid-transfer* :: *account* $\Rightarrow$ *transfer-amount* $\Rightarrow$ *bool* **where**
valid-transfer α τ = (*strictly-solvent* τ $\wedge$ *strictly-solvent* (α $-$ τ))

lemma *valid-transfer-alt-def*: *valid-transfer* α τ = ($0 \leq \tau \wedge \tau \leq \alpha$)
⟨*proof*⟩

Only strictly solvent accounts can make valid transfers to begin with.

lemma *only-strictly-solvent-accounts-can-transfer*:
assumes *valid-transfer* α τ
shows *strictly-solvent* α
⟨*proof*⟩

We may now give a key result: accounts remain strictly solvent given a valid transfer.

theorem *strictly-solvent-still-strictly-solvent-after-valid-transfer*:
assumes *valid-transfer* ($\mathcal{L}$ *a*) τ
and *strictly-solvent* ($\mathcal{L}$ *b*)
shows
strictly-solvent ((*transfer* $\mathcal{L}$ τ *a b*) *a*)
strictly-solvent ((*transfer* $\mathcal{L}$ τ *a b*) *b*)
⟨*proof*⟩

4.4 Embedding Conventional Cash-Only Ledgers

We show that in a sense the ledgers presented generalize conventional ledgers which only track cash.

An account consisting of just cash is strictly solvent if and only if it consists of a non-negative amount of cash.

lemma *strictly-solvent-just-cash-equiv*:
strictly-solvent (*just-cash c*) = ($0 \leq c$)
⟨*proof*⟩

An empty account corresponds to *0*; the account with no cash or debit or credit.

lemma *zero-account-alt-def*: *just-cash 0* = *0*
⟨*proof*⟩

Building on *just-cash 0* = *0*, we have that *just-cash* is an embedding into an ordered subgroup. This means that *just-cash* is an order-preserving group homomorphism from the reals to the universe of accounts.

lemma *just-cash-embed*: ($a = b$) = (*just-cash a* = *just-cash b*)
⟨*proof*⟩

lemma *partial-nav-just-cash* [*simp*]:
($\sum$ $i \leq n$. π (*just-cash a*) *i*) = *a*
⟨*proof*⟩

lemma *just-cash-order-embed*: $(a \leq b) = (\mathit{just\text{-}cash}\ a \leq \mathit{just\text{-}cash}\ b)$
⟨*proof*⟩

lemma *just-cash-plus* [*simp*]: $\mathit{just\text{-}cash}\ a + \mathit{just\text{-}cash}\ b = \mathit{just\text{-}cash}\ (a + b)$
⟨*proof*⟩

lemma *just-cash-uminus* [*simp*]: $- \mathit{just\text{-}cash}\ a = \mathit{just\text{-}cash}\ (- a)$
⟨*proof*⟩

lemma *just-cash-subtract* [*simp*]:
$\mathit{just\text{-}cash}\ a - \mathit{just\text{-}cash}\ b = \mathit{just\text{-}cash}\ (a - b)$
⟨*proof*⟩

Valid transfers as per $\mathit{valid\text{-}transfer}\ ?\alpha\ ?\tau = (0 \leq ?\tau \wedge ?\tau \leq ?\alpha)$ collapse into inequalities over the real numbers.

lemma *just-cash-valid-transfer*:
$\mathit{valid\text{-}transfer}\ (\mathit{just\text{-}cash}\ c)\ (\mathit{just\text{-}cash}\ t) = ((0 :: \mathit{real}) \leq t \wedge t \leq c)$
⟨*proof*⟩

Finally a ledger consisting of accounts with only cash is trivially *balanced*.

lemma (**in** *finite*) *just-cash-summation*:
fixes $A :: 'a\ \mathit{set}$
assumes $\forall\ a \in A.\ \exists\ c\ .\ \mathcal{L}\ a = \mathit{just\text{-}cash}\ c$
shows $\exists\ c\ .\ (\sum\ a \in A\ .\ \mathcal{L}\ a) = \mathit{just\text{-}cash}\ c$
⟨*proof*⟩

lemma (**in** *finite*) *just-cash-UNIV-is-balanced*:
assumes $\forall\ a\ .\ \exists\ c\ .\ \mathcal{L}\ a = \mathit{just\text{-}cash}\ c$
shows $\exists\ c\ .\ \mathit{balanced}\ \mathcal{L}\ c$
⟨*proof*⟩

5 Interest

In this section we discuss how to calculate the interest accrued by an account for a period. This is done by looking at the sum of all of the credit and debit in an account. This sum is called the *net asset value* of an account.

5.1 Net Asset Value

The net asset value of an account is the sum of all of the non-zero entries. Since accounts have finite support this sum is always well defined.

definition $\mathit{net\text{-}asset\text{-}value} :: \mathit{account} \Rightarrow \mathit{real}$ **where**
$\mathit{net\text{-}asset\text{-}value}\ \alpha = (\sum\ i \mid \pi\ \alpha\ i \neq 0\ .\ \pi\ \alpha\ i)$

5.1.1 The Shortest Period for Credited & Debited Assets in an Account

Higher indexes for an account correspond to shorter loan periods. Since accounts only have a finite number of entries, it makes sense to talk about the *shortest* period an account has an entry for. The net asset value for an account has a simpler expression in terms of that account's shortest period.

definition *shortest-period* :: *account* ⇒ *nat* **where**
  shortest-period α =
    (*if* ($\forall$ *i*. π α *i* = *0*)
     then 0
     else Max {*i* . π α *i* $\neq$ *0*})

lemma *shortest-period-uminus*:
  shortest-period ($-$ α) = *shortest-period* α
  ⟨*proof*⟩

lemma *finite-account-support*:
  finite {*i* . π α *i* $\neq$ *0*}
⟨*proof*⟩

lemma *shortest-period-plus*:
  shortest-period (α + β) $\leq$ *max* (*shortest-period* α) (*shortest-period* β)
  (**is** - $\leq$ *?MAX*)
⟨*proof*⟩

lemma *shortest-period-π*:
  assumes π α *i* $\neq$ *0*
  shows π α (*shortest-period* α) $\neq$ *0*
⟨*proof*⟩

lemma *shortest-period-bound*:
  assumes π α *i* $\neq$ *0*
  shows *i* $\leq$ *shortest-period* α
⟨*proof*⟩

Using *shortest-period* we may give an alternate definition for *net-asset-value*.

lemma *net-asset-value-alt-def*:
  net-asset-value α = ($\sum$ *i* $\leq$ *shortest-period* α. π α *i*)
⟨*proof*⟩

lemma *greater-than-shortest-period-zero*:
  assumes *shortest-period* α < *m*
  shows π α *m* = *0*
⟨*proof*⟩

An account's *net-asset-value* does not change when summing beyond its *shortest-period*. This is helpful when computing aggregate net asset values across multiple accounts.

lemma *net-asset-value-shortest-period-ge*:
  assumes *shortest-period* $\alpha \leq n$
  shows *net-asset-value* $\alpha = (\sum\ i \leq n.\ \pi\ \alpha\ i)$
⟨*proof*⟩

5.1.2 Net Asset Value Properties

In this section we explore how *net-asset-value* forms an order-preserving group homomorphism from the universe of accounts to the real numbers.

We first observe that *strictly-solvent* implies the more conventional notion of solvent, where an account's net asset value is non-negative.

lemma *strictly-solvent-net-asset-value*:
  assumes *strictly-solvent* α
  shows $0 \leq$ *net-asset-value* α
  ⟨*proof*⟩

Next we observe that *net-asset-value* is a order preserving group homomorphism from the universe of accounts to *real*.

lemma *net-asset-value-zero* [*simp*]: *net-asset-value* $0 = 0$
  ⟨*proof*⟩

lemma *net-asset-value-mono*:
  assumes $\alpha \leq \beta$
  shows *net-asset-value* $\alpha \leq$ *net-asset-value* β
⟨*proof*⟩

lemma *net-asset-value-uminus*: *net-asset-value* $(-\ \alpha) = -$ *net-asset-value* α
  ⟨*proof*⟩

lemma *net-asset-value-plus*:
  net-asset-value $(\alpha + \beta) =$ *net-asset-value* $\alpha +$ *net-asset-value* β
  (**is** $?lhs = ?\Sigma\alpha + ?\Sigma\beta$)
⟨*proof*⟩

lemma *net-asset-value-minus*:
  net-asset-value $(\alpha - \beta) =$ *net-asset-value* $\alpha -$ *net-asset-value* β
  ⟨*proof*⟩

Finally we observe that *just-cash* is the right inverse of *net-asset-value*.

lemma *net-asset-value-just-cash-left-inverse*:
  net-asset-value (*just-cash* c) $= c$
  ⟨*proof*⟩

5.2 Distributing Interest

We next show that the total interest accrued for a ledger at distribution does not change when one account makes a transfer to another.

definition (**in** *finite*) *total-interest* :: $'a$ *ledger* $\Rightarrow$ *real* $\Rightarrow$ *real*
where *total-interest* $\mathcal{L}$ $i = (\sum\ a \in UNIV.\ i * \textit{net-asset-value}\ (\mathcal{L}\ a))$

lemma (**in** *finite*) *total-interest-transfer*:
total-interest (*transfer* $\mathcal{L}$ τ a b) i = *total-interest* $\mathcal{L}$ i
(**is** *total-interest* $?\mathcal{L}'$ i = -)
⟨*proof*⟩

6 Update

Periodically the ledger is *updated*. When this happens interest is distributed and loans are returned. Each time loans are returned, a fixed fraction of each loan for each period is returned.

The fixed fraction for returned loans is given by a *rate function*. We denote rate functions with ϱ::*nat* $\Rightarrow$ *real*. In principle this function obeys the rules:

- $\varrho\ 0 = 0$ – Cash is not returned.
- $\forall n.\ \varrho\ n < 1$ – The fraction of a loan returned never exceeds 1.
- $\forall n\ m.\ n < m \longrightarrow \varrho\ n < \varrho\ m$ – Higher indexes correspond to shorter loan periods. This in turn corresponds to a higher fraction of loans returned at update for higher indexes.

In practice, rate functions determine the time it takes for 99% of the loan to be returned. However, the presentation here abstracts away from time. In §7.3 we establish a closed form for updating. This permits for a production implementation to efficiently (albeit *lazily*) update ever *millisecond* if so desired.

definition *return-loans* :: (*nat* $\Rightarrow$ *real*) $\Rightarrow$ *account* $\Rightarrow$ *account* **where**
return-loans ϱ α = ι ($\lambda\ n\ .\ (1 - \varrho\ n) * \pi\ \alpha\ n$)

lemma *Rep-account-return-loans* [*simp*]:
π (*return-loans* ϱ α) = ($\lambda\ n\ .\ (1 - \varrho\ n) * \pi\ \alpha\ n$)
⟨*proof*⟩

As discussed, updating an account involves distributing interest and returning its credited and debited loans.

definition *update-account* :: (*nat* $\Rightarrow$ *real*) $\Rightarrow$ *real* $\Rightarrow$ *account* $\Rightarrow$ *account* **where**
update-account ϱ i α = *just-cash* (i * *net-asset-value* α) + *return-loans* ϱ α

definition *update-ledger* :: (*nat* $\Rightarrow$ *real*) $\Rightarrow$ *real* $\Rightarrow$ $'a$ *ledger* $\Rightarrow$ $'a$ *ledger*
where
update-ledger ϱ i $\mathcal{L}$ a = *update-account* ϱ i ($\mathcal{L}$ a)

6.1 Update Preserves Ledger Balance

A key theorem is that if all credit and debit in a ledger cancel, they will continue to cancel after update. In this sense the monetary supply grows with the interest rate, but is otherwise conserved.

A consequence of this theorem is that while counter-party obligations are not explicitly tracked by the ledger, these obligations are fulfilled as funds are returned by the protocol.

definition *shortest-ledger-period* :: *'a ledger* $\Rightarrow$ *nat* **where**
shortest-ledger-period $\mathcal{L}$ = *Max* (*image shortest-period* (*range* $\mathcal{L}$))

lemma (**in** *finite*) *shortest-ledger-period-bound*:
fixes $\mathcal{L}$ :: *'a ledger*
shows *shortest-period* ($\mathcal{L}$ *a*) $\leq$ *shortest-ledger-period* $\mathcal{L}$
⟨*proof*⟩

theorem (**in** *finite*) *update-balanced*:
assumes ϱ *0* = *0* **and** $\forall$ *n*. ϱ *n* < *1* **and** *0* $\leq$ *i*
shows *balanced* $\mathcal{L}$ *c* = *balanced* (*update-ledger* ϱ *i* $\mathcal{L}$) ((*1* + *i*) $*$ *c*)
(**is** - = *balanced* *?*$\mathcal{L}'$ ((*1* + *i*) $*$ *c*))
⟨*proof*⟩

6.2 Strictly Solvent is Forever Strictly Solvent

The final theorem presented in this section is that if an account is strictly solvent, it will still be strictly solvent after update.

This theorem is the key to how the system avoids counter party risk. Provided the system enforces that all accounts are strictly solvent and transfers are *valid* (as discussed in §4.2), all accounts will remain strictly solvent forever.

We first prove that *return-loans* is a group homomorphism.

It is order preserving given certain assumptions.

lemma *return-loans-plus*:
return-loans ϱ (α + β) = *return-loans* ϱ α + *return-loans* ϱ β
⟨*proof*⟩

lemma *return-loans-zero* [*simp*]: *return-loans* ϱ *0* = *0*
⟨*proof*⟩

lemma *return-loans-uminus*: *return-loans* ϱ ($-$ α) = $-$ *return-loans* ϱ α
⟨*proof*⟩

lemma *return-loans-subtract*:
return-loans ϱ (α $-$ β) = *return-loans* ϱ α $-$ *return-loans* ϱ β

$\langle proof \rangle$

As presented in §1, each index corresponds to a progressively shorter loan period. This is captured by a monotonicity assumption on the rate function ϱ::*nat* $\Rightarrow$ *real*. In particular, provided $\forall n.\ \varrho\ n < 1$ and $\forall n\ m.\ n < m \longrightarrow \varrho\ n < \varrho\ m$ then we know that all outstanding credit is going away faster than loans debited for longer periods.

Given the monotonicity assumptions for a rate function ϱ::*nat* $\Rightarrow$ *real*, we may in turn prove monotonicity for *return-loans* over $(\leq)$::*account* $\Rightarrow$ *account* $\Rightarrow$ *bool*.

lemma *return-loans-mono*:
assumes $\forall\ n\ .\ \varrho\ n < 1$
and $\forall\ n\ m\ .\ n \leq m \longrightarrow \varrho\ n \leq \varrho\ m$
and $\alpha \leq \beta$
shows *return-loans* $\varrho\ \alpha \leq$ *return-loans* $\varrho\ \beta$
$\langle proof \rangle$

lemma *return-loans-just-cash*:
assumes $\varrho\ 0 = 0$
shows *return-loans* ϱ (*just-cash* c) = *just-cash* c
$\langle proof \rangle$

lemma *distribute-interest-plus*:
just-cash (i * *net-asset-value* $(\alpha + \beta)$) =
just-cash (i * *net-asset-value* α) +
just-cash (i * *net-asset-value* β)
$\langle proof \rangle$

We now prove that *update-account* is an order-preserving group homomorphism just as *just-cash*, *net-asset-value*, and *return-loans* are.

lemma *update-account-plus*:
update-account $\varrho\ i\ (\alpha + \beta)$ =
update-account $\varrho\ i\ \alpha$ + *update-account* $\varrho\ i\ \beta$
$\langle proof \rangle$

lemma *update-account-zero* [*simp*]: *update-account* $\varrho\ i\ 0 = 0$
$\langle proof \rangle$

lemma *update-account-uminus*:
update-account $\varrho\ i\ (-\alpha) = -$ *update-account* $\varrho\ i\ \alpha$
$\langle proof \rangle$

lemma *update-account-subtract*:
update-account $\varrho\ i\ (\alpha - \beta)$ =
update-account $\varrho\ i\ \alpha$ − *update-account* $\varrho\ i\ \beta$
$\langle proof \rangle$

lemma *update-account-mono*:
assumes $0 \leq i$
and $\forall\ n\ .\ \varrho\ n < 1$
and $\forall\ n\ m\ .\ n \leq m \longrightarrow \varrho\ n \leq \varrho\ m$
and $\alpha \leq \beta$
shows $\textit{update-account}\ \varrho\ i\ \alpha \leq \textit{update-account}\ \varrho\ i\ \beta$
⟨*proof*⟩

It follows from monotonicity and $\textit{update-account}\ \varrho\ i\ 0 = 0$ that strictly solvent accounts remain strictly solvent after update.

lemma *update-preserves-strictly-solvent*:
assumes $0 \leq i$
and $\forall\ n\ .\ \varrho\ n < 1$
and $\forall\ n\ m\ .\ n \leq m \longrightarrow \varrho\ n \leq \varrho\ m$
and $\textit{strictly-solvent}\ \alpha$
shows $\textit{strictly-solvent}\ (\textit{update-account}\ \varrho\ i\ \alpha)$
⟨*proof*⟩

7 Bulk Update

In this section we demonstrate there exists a closed form for bulk-updating an account.

primrec *bulk-update-account* ::
$\textit{nat} \Rightarrow (\textit{nat} \Rightarrow \textit{real}) \Rightarrow \textit{real} \Rightarrow \textit{account} \Rightarrow \textit{account}$
where
$\textit{bulk-update-account}\ 0\ \text{-}\ \text{-}\ \alpha = \alpha$
| $\textit{bulk-update-account}\ (\textit{Suc}\ n)\ \varrho\ i\ \alpha =$
$\textit{update-account}\ \varrho\ i\ (\textit{bulk-update-account}\ n\ \varrho\ i\ \alpha)$

As with *update-account*, *bulk-update-account* is an order-preserving group homomorphism.

We now prove that *update-account* is an order-preserving group homomorphism just as *just-cash*, *net-asset-value*, and *return-loans* are.

lemma *bulk-update-account-plus*:
$\textit{bulk-update-account}\ n\ \varrho\ i\ (\alpha + \beta) =$
$\textit{bulk-update-account}\ n\ \varrho\ i\ \alpha + \textit{bulk-update-account}\ n\ \varrho\ i\ \beta$
⟨*proof*⟩

lemma *bulk-update-account-zero* [*simp*]: $\textit{bulk-update-account}\ n\ \varrho\ i\ 0 = 0$
⟨*proof*⟩

lemma *bulk-update-account-uminus*:
$\textit{bulk-update-account}\ n\ \varrho\ i\ (-\alpha) = -\ \textit{bulk-update-account}\ n\ \varrho\ i\ \alpha$
⟨*proof*⟩

lemma *bulk-update-account-subtract*:
$\mathit{bulk\text{-}update\text{-}account}\ n\ \varrho\ i\ (\alpha - \beta) =$
$\mathit{bulk\text{-}update\text{-}account}\ n\ \varrho\ i\ \alpha - \mathit{bulk\text{-}update\text{-}account}\ n\ \varrho\ i\ \beta$
⟨*proof*⟩

lemma *bulk-update-account-mono*:
assumes $0 \leq i$
and $\forall\ n\ .\ \varrho\ n < 1$
and $\forall\ n\ m\ .\ n \leq m \longrightarrow \varrho\ n \leq \varrho\ m$
and $\alpha \leq \beta$
shows $\mathit{bulk\text{-}update\text{-}account}\ n\ \varrho\ i\ \alpha \leq \mathit{bulk\text{-}update\text{-}account}\ n\ \varrho\ i\ \beta$
⟨*proof*⟩

In follows from the fact that *bulk-update-account* is an order-preserving group homomorphism that the update protocol is *safe*. Informally this means that provided we enforce every account is strictly solvent then no account will ever have negative net asset value (ie, be in the red).

theorem *bulk-update-safety*:
assumes $0 \leq i$
and $\forall\ n\ .\ \varrho\ n < 1$
and $\forall\ n\ m\ .\ n \leq m \longrightarrow \varrho\ n \leq \varrho\ m$
and *strictly-solvent* α
shows $0 \leq \mathit{net\text{-}asset\text{-}value}\ (\mathit{bulk\text{-}update\text{-}account}\ n\ \varrho\ i\ \alpha)$
⟨*proof*⟩

7.1 Decomposition

In order to express *bulk-update-account* using a closed formulation, we first demonstrate how to *decompose* an account into a summation of credited and debited loans for different periods.

definition *loan* :: *nat* ⇒ *real* ⇒ *account* (‹δ›)
where
$\delta\ n\ x = \iota\ (\lambda\ m\ .\ \mathit{if}\ n = m\ \mathit{then}\ x\ \mathit{else}\ 0)$

lemma *loan-just-cash*: $\delta\ 0\ c = \mathit{just\text{-}cash}\ c$
⟨*proof*⟩

lemma *Rep-account-loan* [*simp*]:
$\pi\ (\delta\ n\ x) = (\lambda\ m\ .\ \mathit{if}\ n = m\ \mathit{then}\ x\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *loan-zero* [*simp*]: $\delta\ n\ 0 = 0$
⟨*proof*⟩

lemma *shortest-period-loan*:
assumes $c \neq 0$
shows $\mathit{shortest\text{-}period}\ (\delta\ n\ c) = n$
⟨*proof*⟩

lemma *net-asset-value-loan* [*simp*]: *net-asset-value* (δ *n c*) = *c*
⟨*proof*⟩

lemma *return-loans-loan* [*simp*]: *return-loans* ϱ (δ *n c*) = δ *n* ((*1* − ϱ *n*) ∗ *c*)
⟨*proof*⟩

lemma *account-decomposition*:
$\alpha = (\sum\ i \leq \textit{shortest-period}\ \alpha.\ \delta\ i\ (\pi\ \alpha\ i))$
⟨*proof*⟩

7.2 Simple Transfers

Building on our decomposition, we can understand the necessary and sufficient conditions to transfer a loan of $\delta\ n\ c$.

We first give a notion of the *reserves for a period n*. This characterizes the available funds for a loan of period n that an account α possesses.

definition *reserves-for-period* :: *account* ⇒ *nat* ⇒ *real* **where**
  reserves-for-period α *n* =
    fold
      min
      $[(\sum\ i{\leq}k\ .\ \pi\ \alpha\ i)\ .\ k \leftarrow [n..{<}\textit{shortest-period}\ \alpha{+}1]]$
      $(\sum\ i{\leq}n\ .\ \pi\ \alpha\ i)$

lemma *nav-reserves-for-period*:
  assumes *shortest-period* $\alpha \leq n$
  shows *reserves-for-period* α *n* = *net-asset-value* α
⟨*proof*⟩

lemma *reserves-for-period-exists*:
  $\exists\, m{\geq}n.\ \textit{reserves-for-period}\ \alpha\ n = (\sum\ i{\leq}m\ .\ \pi\ \alpha\ i)$
    $\wedge\ (\forall\, u{\geq}n.\ (\sum\ i{\leq}m\ .\ \pi\ \alpha\ i) \leq (\sum\ i{\leq}u\ .\ \pi\ \alpha\ i))$
⟨*proof*⟩

lemma *permissible-loan-converse*:
  assumes *strictly-solvent* (α − δ *n c*)
  shows *c* ≤ *reserves-for-period* α *n*
⟨*proof*⟩

lemma *permissible-loan*:
  assumes *strictly-solvent* α
  shows *strictly-solvent* (α − δ *n c*) = (*c* ≤ *reserves-for-period* α *n*)
⟨*proof*⟩

7.3 Closed Forms

We first give closed forms for loans $\delta\ n\ c$. The simplest closed form is for *just-cash*. Here the closed form is just the compound interest accrued from each update.

lemma *bulk-update-just-cash-closed-form*:
  assumes $\varrho\ 0 = 0$
  shows $\textit{bulk-update-account}\ n\ \varrho\ i\ (\textit{just-cash}\ c) =$
    $\textit{just-cash}\ ((1 + i)\ \hat{}\ n * c)$
⟨*proof*⟩

lemma *bulk-update-loan-closed-form*:
  assumes $\varrho\ k \neq 1$
  and $\varrho\ k > 0$
  and $\varrho\ 0 = 0$
  and $i \geq 0$
  shows $\textit{bulk-update-account}\ n\ \varrho\ i\ (\delta\ k\ c) =$
    $\textit{just-cash}\ (c * i * ((1 + i)\ \hat{}\ n - (1 - \varrho\ k)\ \hat{}\ n) / (i + \varrho\ k))$
    $+ \delta\ k\ ((1 - \varrho\ k)\ \hat{}\ n * c)$
⟨*proof*⟩

We next give an *algebraic* closed form. This uses the ordered abelian group that *account*s form.

lemma *bulk-update-algebraic-closed-form*:
  assumes $0 \leq i$
  and $\forall\ n\ .\ \varrho\ n < 1$
  and $\forall\ n\ m\ .\ n < m \longrightarrow \varrho\ n < \varrho\ m$
  and $\varrho\ 0 = 0$
  shows $\textit{bulk-update-account}\ n\ \varrho\ i\ \alpha$
    $= \textit{just-cash}\ ($
      $(1 + i)\ \hat{}\ n * (\textit{cash-reserve}\ \alpha)$
      $+ (\sum\ k = 1..\textit{shortest-period}\ \alpha.$
        $(\pi\ \alpha\ k) * i * ((1 + i)\ \hat{}\ n - (1 - \varrho\ k)\ \hat{}\ n)$
          $/ (i + \varrho\ k))$
    $)$
    $+ (\sum k = 1..\textit{shortest-period}\ \alpha.\ \delta\ k\ ((1 - \varrho\ k)\ \hat{}\ n * \pi\ \alpha\ k))$
⟨*proof*⟩

We finally give a *functional* closed form for bulk updating an account. Since the form is in terms of exponentiation, we may efficiently compute the bulk update output using *exponentiation-by-squaring*.

theorem *bulk-update-closed-form*:
  assumes $0 \leq i$
  and $\forall\ n\ .\ \varrho\ n < 1$
  and $\forall\ n\ m\ .\ n < m \longrightarrow \varrho\ n < \varrho\ m$
  and $\varrho\ 0 = 0$
  shows $\textit{bulk-update-account}\ n\ \varrho\ i\ \alpha$
    $= \iota\ (\ \lambda\ k\ .$
      $\textit{if}\ k = 0\ \textit{then}$
        $(1 + i)\ \hat{}\ n * (\textit{cash-reserve}\ \alpha)$
        $+ (\sum\ j = 1..\textit{shortest-period}\ \alpha.$
          $(\pi\ \alpha\ j) * i * ((1 + i)\ \hat{}\ n - (1 - \varrho\ j)\ \hat{}\ n)$

$/ \ (i + \varrho\ j))$

else

$(1 - \varrho\ k)\ \hat{}\ n * \pi\ \alpha\ k$

$)$

(**is** - $= \iota\ ?\nu$)

$\langle proof \rangle$

end