

Risk-Free Lending

Matthew Doty

March 8, 2026

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

1	Accounts	1
2	Strictly Solvent	4
3	Cash	6
4	Ledgers	7
4.1	Balanced Ledgers	8
4.2	Transfers	9
4.3	The Valid Transfers Protocol	11
4.4	Embedding Conventional Cash-Only Ledgers	12
5	Interest	14
5.1	Net Asset Value	14
5.1.1	The Shortest Period for Credited & Debited Assets in an Account	15

5.1.2	Net Asset Value Properties	19
5.2	Distributing Interest	21
6	Update	22
6.1	Update Preserves Ledger Balance	23
6.2	Strictly Solvent is Forever Strictly Solvent	28
7	Bulk Update	32
7.1	Decomposition	34
7.2	Simple Transfers	36
7.3	Closed Forms	43
theory	<i>Risk-Free-Lending</i>	
imports		
	<i>Complex-Main</i>	
	<i>HOL-Cardinals.Cardinals</i>	
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  $\Rightarrow$  real) set
proof –
  have ( $\lambda$  . 0)  $\in$  fin-support 0 UNIV
    unfolding fin-support-def support-def
    by simp
  thus  $\exists x :: nat \Rightarrow real. x \in fin-support 0 UNIV$  by fastforce

```

qed

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* ($\langle \pi \rangle$)

notation *Abs-account* ($\langle \iota \rangle$)

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 $0 = \iota (\lambda \cdot . 0)$

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 $(\alpha_1 :: \textit{account}) - \alpha_2 = \alpha_1 + - \alpha_2$

lemma *Rep-account-zero* [*simp*]: $\pi 0 = (\lambda \cdot . 0)$

proof –

have $(\lambda \cdot . 0) \in \textit{fin-support } 0 \textit{ UNIV}$

unfolding *fin-support-def support-def*

by *simp*

thus *?thesis*

unfolding *zero-account-def*

using *Abs-account-inverse* **by** *blast*

qed

lemma *Rep-account-uminus* [*simp*]:

$\pi (- \alpha) = (\lambda n . - \pi \alpha n)$

proof –

have $\pi \alpha \in \textit{fin-support } 0 \textit{ UNIV}$

using *Rep-account* **by** *blast*

hence $(\lambda x . - \pi \alpha x) \in \textit{fin-support } 0 \textit{ UNIV}$

unfolding *fin-support-def support-def*

by *force*

thus *?thesis*

unfolding *uminus-account-def*

using *Abs-account-inverse* **by** *blast*

qed

lemma *fin-support-closed-under-addition*:

fixes $f g :: 'a \Rightarrow \textit{real}$

```

assumes  $f \in \text{fin-support } 0 \ A$ 
and  $g \in \text{fin-support } 0 \ A$ 
shows  $(\lambda x . f x + g x) \in \text{fin-support } 0 \ A$ 
using assms
unfolding fin-support-def support-def
by (metis (mono-tags) mem-Collect-eq sum.finite-Collect-op)

lemma Rep-account-plus [simp]:
 $\pi (\alpha_1 + \alpha_2) = (\lambda n . \pi \alpha_1 n + \pi \alpha_2 n)$ 
unfolding plus-account-def
by (metis (full-types)
      Abs-account-cases
      Abs-account-inverse
      fin-support-closed-under-addition)

instance
proof(standard)
  fix  $a \ b \ c :: \text{account}$ 
  have  $\forall n . \pi (a + b) n + \pi c n = \pi a n + \pi (b + c) n$ 
    using Rep-account-plus plus-account-def
    by auto
  thus  $a + b + c = a + (b + c)$ 
    unfolding plus-account-def
    by force
next
  fix  $a \ b :: \text{account}$ 
  show  $a + b = b + a$ 
    unfolding plus-account-def
    by (simp add: add.commute)
next
  fix  $a :: \text{account}$ 
  show  $0 + a = a$ 
    unfolding plus-account-def Rep-account-zero
    by (simp add: Rep-account-inverse)
next
  fix  $a :: \text{account}$ 
  show  $- a + a = 0$ 
    unfolding plus-account-def zero-account-def Rep-account-uminus
    by (simp add: Abs-account-inverse)
next
  fix  $a \ b :: \text{account}$ 
  show  $a - b = a + - b$ 
    using minus-account-def by blast
qed

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* $\alpha \implies 0 \leq \text{net-asset-value } \alpha$ in §5.1.2.

definition *strictly-solvent* :: *account* \Rightarrow *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* $(\alpha_1 + \alpha_2)$
using *assms Rep-account-plus*
unfolding *strictly-solvent-def plus-account-def*
by (*simp add: Abs-account-inverse sum.distrib*)

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* \Rightarrow *account* \Rightarrow *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* \Rightarrow *account* \Rightarrow *bool* **where**
less-account $\alpha_1 \alpha_2 \equiv (\alpha_1 \leq \alpha_2 \wedge \neg \alpha_2 \leq \alpha_1)$

instance

proof(*standard*)

fix $x y :: \text{account}$
show $(x < y) = (x \leq y \wedge \neg y \leq x)$
unfolding *less-account-def* ..

next

fix $x :: \text{account}$
show $x \leq x$
unfolding *less-eq-account-def* **by** *auto*

next

fix $x y z :: \text{account}$
assume $x \leq y$ **and** $y \leq z$
thus $x \leq z$
unfolding *less-eq-account-def*
by (*meson order-trans*)

next

fix $x y :: \text{account}$
assume $x \leq y$ **and** $y \leq x$
hence $\star: \forall n . (\sum_{i \leq n} \pi x i) = (\sum_{i \leq n} \pi y i)$
unfolding *less-eq-account-def*

```

using dual-order.antisym by blast
{
  fix n
  have  $\pi x n = \pi y n$ 
  proof (cases n = 0)
    case True
    then show ?thesis using  $\star$ 
      by (metis
          atMost-0
          empty-iff
          finite.intros(1)
          group-cancel.rule0
          sum.empty sum.insert)
  next
  case False
  from this obtain m where
    n = m + 1
    by (metis Nat.add-0-right Suc-eq-plus1 add-eq-iff)
  have  $(\sum_{i \leq n} \pi x i) = (\sum_{i \leq n} \pi y i)$ 
    using  $\star$  by auto
  hence
     $(\sum_{i \leq m} \pi x i) + \pi x n =$ 
     $(\sum_{i \leq m} \pi y i) + \pi y n$ 
    using  $\langle n = m + 1 \rangle$ 
    by simp
  moreover have  $(\sum_{i \leq m} \pi x i) = (\sum_{i \leq m} \pi y i)$ 
    using  $\star$  by auto
  ultimately show ?thesis by linarith
qed
}
hence  $\pi x = \pi y$  by auto
thus  $x = y$ 
  by (metis Rep-account-inverse)
next
fix x y z :: account
assume  $x \leq y$ 
{
  fix n :: nat
  have
     $(\sum_{i \leq n} \pi (z + x) i) =$ 
     $(\sum_{i \leq n} \pi z i) + (\sum_{i \leq n} \pi x i)$ 
  and
     $(\sum_{i \leq n} \pi (z + y) i) =$ 
     $(\sum_{i \leq n} \pi z i) + (\sum_{i \leq n} \pi y i)$ 
  unfolding Rep-account-plus
  by (simp add: sum.distrib)+
  moreover have  $(\sum_{i \leq n} \pi x i) \leq (\sum_{i \leq n} \pi y i)$ 
    using  $\langle x \leq y \rangle$ 
    unfolding less-eq-account-def by blast
}

```

```

ultimately have
  ( $\sum_{i \leq n} \pi (z + x) i$ )  $\leq$  ( $\sum_{i \leq n} \pi (z + y) i$ )
  by linarith
}
thus  $z + x \leq z + y$ 
  unfolding
    less-eq-account-def by auto
qed
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  $\alpha = (0 \leq \alpha)$ 
  unfolding
    strictly-solvent-def
    less-eq-account-def
  using zero-account-def
  by force

```

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  $\Rightarrow$  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  $\alpha$ 
  shows  $0 \leq \text{cash-reserve } \alpha$ 
  using assms
  unfolding strictly-solvent-def cash-reserve-def
  by (metis
    atMost-0
    empty-iff
    finite.emptyI
    group-cancel.rule0
    sum.empty
    sum.insert)

```

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

```

definition just-cash :: real  $\Rightarrow$  account where
  just-cash  $c = \iota (\lambda n . \text{if } n = 0 \text{ then } c \text{ else } 0)$ 

```

```

lemma Rep-account-just-cash [simp]:
   $\pi$  (just-cash  $c$ ) = ( $\lambda n . \text{if } n = 0 \text{ then } c \text{ else } 0$ )
proof (cases  $c = 0$ )
  case True
  hence just-cash  $c = 0$ 
    unfolding just-cash-def zero-account-def
    by force
  then show ?thesis
    using Rep-account-zero True by force
next
  case False
  hence finite (support 0 UNIV ( $\lambda n :: \text{nat} . \text{if } n = 0 \text{ then } c \text{ else } 0$ ))
    unfolding support-def
    by auto
  hence ( $\lambda n :: \text{nat} . \text{if } n = 0 \text{ then } c \text{ else } 0$ )  $\in$  fin-support 0 UNIV
    unfolding fin-support-def
    by blast
  then show ?thesis
    unfolding just-cash-def
    using Abs-account-inverse by auto
qed

```

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* = $'a \Rightarrow \text{account}$

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} $c \equiv (\sum a \in \text{UNIV}. \mathcal{L} a) = \text{just-cash } c$

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-solvent* ($\sum a \in \text{UNIV}. \mathcal{L} a$)
using *assms*
unfolding *balanced-def strictly-solvent-def*
by *simp*

lemma (in *finite*) *finite-Rep-account-ledger* [*simp*]:
 $\pi (\sum a \in (A :: 'a \text{ set}). \mathcal{L} a) n = (\sum a \in A. \pi (\mathcal{L} a) n)$
using *finite*
by (*induct A rule: finite-induct, auto*)

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 $\mathcal{L} c =$
 $((\sum a \in UNIV. \text{cash-reserve } (\mathcal{L} a)) = c$
 $\wedge (\forall n > 0. (\sum a \in UNIV. \pi (\mathcal{L} a) n) = 0))$
(is *?lhs = ?rhs*
proof (*rule iffI*)
assume *?lhs*
hence $(\sum a \in UNIV. \text{cash-reserve } (\mathcal{L} a)) = c$
unfolding *balanced-def cash-reserve-def*
by (*metis Rep-account-just-cash finite-Rep-account-ledger*)
moreover
{
fix *n :: nat*
assume $n > 0$
with $\langle ?lhs \rangle$ **have** $(\sum a \in UNIV. \pi (\mathcal{L} a) n) = 0$
unfolding *balanced-def*
by (*metis*
Rep-account-just-cash
less-nat-zero-code
finite-Rep-account-ledger)
}
ultimately show *?rhs* **by** *auto*
next
assume *?rhs*
have $\text{cash-reserve } (\text{just-cash } c) = c$
unfolding *cash-reserve-def*
using *Rep-account-just-cash*
by *presburger*
also have $\dots = (\sum a \in UNIV. \text{cash-reserve } (\mathcal{L} a))$ **using** $\langle ?rhs \rangle$ **by** *auto*
finally have
 $\text{cash-reserve } (\sum a \in UNIV. \mathcal{L} a) = \text{cash-reserve } (\text{just-cash } c)$
unfolding *cash-reserve-def*
by *auto*
moreover
{
fix *n :: nat*
assume $n > 0$
hence $\pi (\sum a \in UNIV. \mathcal{L} a) n = 0$ **using** $\langle ?rhs \rangle$ **by** *auto*
hence $\pi (\sum a \in UNIV. \mathcal{L} a) n = \pi (\text{just-cash } c) n$
unfolding *Rep-account-just-cash* **using** $\langle n > 0 \rangle$ **by** *auto*
}
}

ultimately have
 $\forall n . \pi (\sum a \in UNIV . \mathcal{L} a) n = \pi (just-cash\ c) n$
unfolding *cash-reserve-def*
by (*metis gr-zeroI*)
hence $\pi (\sum a \in UNIV . \mathcal{L} a) = \pi (just-cash\ c)$
by *auto*
thus *?lhs*
unfolding *balanced-def*
using *Rep-account-inject*
by *blast*
qed

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**
transfer $\mathcal{L} \tau a b x =$ (if $a = b$ then $\mathcal{L} x$
else if $x = a$ then $\mathcal{L} a - \tau$
else if $x = b$ then $\mathcal{L} b + \tau$
else $\mathcal{L} x$)

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

lemma *transfer-collapse*:
transfer $\mathcal{L} \tau a a = \mathcal{L}$
unfolding *transfer-def* **by** *auto*

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 $(\sum a \in UNIV . \mathcal{L} a) = (\sum a \in UNIV . transfer\ \mathcal{L} \tau\ x\ y\ a)$
(is $- = (\sum a \in UNIV . ?\mathcal{L}' a)$)
proof (*cases* $x = y$)
case *True*
show *?thesis*
unfolding $\langle x = y \rangle$ *transfer-collapse* ..
next
case *False*
let $?sum-\mathcal{L} = (\sum a \in UNIV - \{x,y\} . \mathcal{L} a)$
let $?sum-\mathcal{L}' = (\sum a \in UNIV - \{x,y\} . ?\mathcal{L}' a)$
have $\forall a \in UNIV - \{x,y\} . ?\mathcal{L}' a = \mathcal{L} a$
by (*simp add: transfer-def*)
hence $?sum-\mathcal{L}' = ?sum-\mathcal{L}$

```

  by (meson sum.cong)
have {x,y} ⊆ UNIV by auto
have (∑ a ∈ UNIV. ?L' a) = ?sum-L' + (∑ a ∈ {x,y}. ?L' a)
  using finite-UNIV sum.subset-diff [OF {x,y} ⊆ UNIV]
  by fastforce
also have ... = ?sum-L' + ?L' x + ?L' y
  using
    ⟨x ≠ y⟩
    finite
    Diff-empty
    Diff-insert-absorb
    Diff-subset
    group-cancel.add1
    insert-absorb
    sum.subset-diff
  by (simp add: insert-Diff-if)
also have ... = ?sum-L' + L x - τ + L y + τ
  unfolding transfer-def
  using ⟨x ≠ y⟩
  by auto
also have ... = ?sum-L' + L x + L y
  by simp
also have ... = ?sum-L + L x + L y
  unfolding ⟨?sum-L' = ?sum-L⟩ ..
also have ... = ?sum-L + (∑ a ∈ {x,y}. L a)
  using
    ⟨x ≠ y⟩
    finite
    Diff-empty
    Diff-insert-absorb
    Diff-subset
    group-cancel.add1
    insert-absorb
    sum.subset-diff
  by (simp add: insert-Diff-if)
ultimately show ?thesis
  by (metis local.finite sum.subset-diff top-greatest)
qed

```

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*:
balanced L c = *balanced* (transfer L τ a b) c
unfolding *balanced-def*
using *sum-transfer-equiv*
by *force*

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 $\text{strictly-solvent } (\sum a \in \text{UNIV}. \mathcal{L}\ a) =$
 $\text{strictly-solvent } (\sum a \in \text{UNIV}. \text{transfer } \mathcal{L}\ \tau\ x\ y\ a)$
using *sum-transfer-equiv*
by *presburger*

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* $:: \text{account} \Rightarrow \text{transfer-amount} \Rightarrow \text{bool}$ **where**
 $\text{valid-transfer } \alpha\ \tau = (\text{strictly-solvent } \tau \wedge \text{strictly-solvent } (\alpha - \tau))$

lemma *valid-transfer-alt-def*: $\text{valid-transfer } \alpha\ \tau = (0 \leq \tau \wedge \tau \leq \alpha)$
unfolding *valid-transfer-def* *strictly-solvent-alt-def*
by *simp*

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

lemma *only-strictly-solvent-accounts-can-transfer*:
assumes $\text{valid-transfer } \alpha\ \tau$
shows $\text{strictly-solvent } \alpha$
using *assms*
unfolding *strictly-solvent-alt-def* *valid-transfer-alt-def*
by *auto*

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 $\text{valid-transfer } (\mathcal{L}\ a)\ \tau$
and $\text{strictly-solvent } (\mathcal{L}\ b)$
shows
 $\text{strictly-solvent } ((\text{transfer } \mathcal{L}\ \tau\ a\ b)\ a)$
 $\text{strictly-solvent } ((\text{transfer } \mathcal{L}\ \tau\ a\ b)\ b)$
using *assms*
unfolding
 $\text{strictly-solvent-alt-def}$
 $\text{valid-transfer-alt-def}$
 transfer-def
by (*cases* $a = b$, *auto*)

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 ≤ c)
unfolding *strictly-solvent-def*
using *Rep-account-just-cash just-cash-def* **by force**

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

lemma *zero-account-alt-def: just-cash 0 = 0*
unfolding *zero-account-def just-cash-def*
by simp

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 (*rule iffI*)
assume $a = b$
thus $just-cash\ a = just-cash\ b$
by force
next
assume $just-cash\ a = just-cash\ b$
hence $cash-reserve\ (just-cash\ a) = cash-reserve\ (just-cash\ b)$
by presburger
thus $a = b$
unfolding *Rep-account-just-cash cash-reserve-def*
by auto
qed

lemma *partial-nav-just-cash [simp]*:
 $(\sum_{i \leq n} . \pi\ (just-cash\ a)\ i) = a$
unfolding *Rep-account-just-cash*
by (induct n, auto)

lemma *just-cash-order-embed: (a ≤ b) = (just-cash a ≤ just-cash b)*
unfolding *less-eq-account-def*
by simp

lemma *just-cash-plus [simp]: just-cash a + just-cash b = just-cash (a + b)*
proof –
{
fix x

```

have  $\pi$  (just-cash  $a + \textit{just-cash } b$ )  $x = \pi$  (just-cash ( $a + b$ ))  $x$ 
proof (cases  $x = 0$ )
  case True
    then show ?thesis
      using Rep-account-just-cash just-cash-def by force
  next
    case False
      then show ?thesis by simp
  qed
}
hence  $\pi$  (just-cash  $a + \textit{just-cash } b$ ) =  $\pi$  (just-cash ( $a + b$ ))
by auto
thus ?thesis
by (metis Rep-account-inverse)
qed

```

```

lemma just-cash-uminus [simp]:  $-\textit{just-cash } a = \textit{just-cash } (- a)$ 
proof -
  {
    fix  $x$ 
    have  $\pi$  ( $-\textit{just-cash } a$ )  $x = \pi$  (just-cash ( $- a$ ))  $x$ 
    proof (cases  $x = 0$ )
      case True
        then show ?thesis
          using Rep-account-just-cash just-cash-def by force
      next
        case False
          then show ?thesis by simp
      qed
    }
  hence  $\pi$  ( $-\textit{just-cash } a$ ) =  $\pi$  (just-cash ( $- a$ ))
  by auto
  thus ?thesis
  by (metis Rep-account-inverse)
qed

```

```

lemma just-cash-subtract [simp]:
   $\textit{just-cash } a - \textit{just-cash } b = \textit{just-cash } (a - b)$ 
  by (simp add: minus-account-def)

```

Valid transfers as per *valid-transfer* $?a \ ?\tau = (0 \leq ?\tau \wedge ?\tau \leq ?a)$ collapse into inequalities over the real numbers.

```

lemma just-cash-valid-transfer:
   $\textit{valid-transfer } (\textit{just-cash } c) (\textit{just-cash } t) = ((0 :: \textit{real}) \leq t \wedge t \leq c)$ 
  unfolding valid-transfer-alt-def
  by (simp add: less-eq-account-def)

```

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

```

lemma (in finite) just-cash-summation:

```

fixes $A :: 'a \text{ set}$
assumes $\forall a \in A. \exists c. \mathcal{L} a = \text{just-cash } c$
shows $\exists c. (\sum a \in A. \mathcal{L} a) = \text{just-cash } c$
using *finite assms*
by (*induct A rule: finite-induct, auto, metis zero-account-alt-def*)

lemma (*in finite just-cash-UNIV-is-balanced:*

assumes $\forall a. \exists c. \mathcal{L} a = \text{just-cash } c$
shows $\exists c. \text{balanced } \mathcal{L} c$
unfolding *balanced-def*
using
assms
just-cash-summation [where A=UNIV]
by *simp*

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 *net-asset-value* $:: \text{account} \Rightarrow \text{real}$ **where**
net-asset-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* $:: \text{account} \Rightarrow \text{nat}$ **where**
shortest-period $\alpha =$
(if $(\forall i. \pi \alpha i = 0)$
then 0
else $\text{Max } \{i . \pi \alpha i \neq 0\}$ *)*

lemma *shortest-period-uminus:*
shortest-period $(- \alpha) = \text{shortest-period } \alpha$
unfolding *shortest-period-def*
using *Rep-account-uminus uminus-account-def*
by *force*

```

lemma finite-account-support:
  finite {i .  $\pi$   $\alpha$  i  $\neq$  0}
proof –
  have  $\pi$   $\alpha$   $\in$  fin-support 0 UNIV
    by (simp add: Rep-account)
  thus ?thesis
    unfolding fin-support-def support-def
    by fastforce
qed

lemma shortest-period-plus:
  shortest-period ( $\alpha$  +  $\beta$ )  $\leq$  max (shortest-period  $\alpha$ ) (shortest-period  $\beta$ )
  (is -  $\leq$  ?MAX)
proof (cases  $\forall$  i .  $\pi$  ( $\alpha$  +  $\beta$ ) i = 0)
  case True
    then show ?thesis unfolding shortest-period-def by auto
  next
    case False
    have shortest-period  $\alpha$   $\leq$  ?MAX and shortest-period  $\beta$   $\leq$  ?MAX
      by auto
    moreover
    have  $\forall$  i > shortest-period  $\alpha$  .  $\pi$   $\alpha$  i = 0
    and  $\forall$  i > shortest-period  $\beta$  .  $\pi$   $\beta$  i = 0
      unfolding shortest-period-def
      using finite-account-support Max.coboundedI leD Collect-cong
      by auto
    ultimately
    have  $\forall$  i > ?MAX .  $\pi$   $\alpha$  i = 0
    and  $\forall$  i > ?MAX .  $\pi$   $\beta$  i = 0
      by simp+
    hence  $\forall$  i > ?MAX .  $\pi$  ( $\alpha$  +  $\beta$ ) i = 0
      by simp
    hence  $\forall$  i .  $\pi$  ( $\alpha$  +  $\beta$ ) i  $\neq$  0  $\longrightarrow$  i  $\leq$  ?MAX
      by (meson not-le)
    thus ?thesis
      unfolding shortest-period-def
      using
        finite-account-support [where  $\alpha$  =  $\alpha$  +  $\beta$ ]
        False
      by simp
qed

lemma shortest-period- $\pi$ :
  assumes  $\pi$   $\alpha$  i  $\neq$  0
  shows  $\pi$   $\alpha$  (shortest-period  $\alpha$ )  $\neq$  0
proof –
  let ?support = {i .  $\pi$   $\alpha$  i  $\neq$  0}
  have A: finite ?support

```

```

    using finite-account-support by blast
  have B: ?support ≠ {} using assms by auto
  have shortest-period α = Max ?support
    using assms
    unfolding shortest-period-def
    by auto
  have shortest-period α ∈ ?support
    unfolding ⟨shortest-period α = Max ?support⟩
    using Max-in [OF A B] by auto
  thus ?thesis
    by auto
qed

```

```

lemma shortest-period-bound:
  assumes π α i ≠ 0
  shows i ≤ shortest-period α
proof -
  let ?support = {i . π α i ≠ 0}
  have shortest-period α = Max ?support
    using assms
    unfolding shortest-period-def
    by auto
  have shortest-period α ∈ ?support
    using assms shortest-period-π by force
  thus ?thesis
    unfolding ⟨shortest-period α = Max ?support⟩
    by (simp add: assms finite-account-support)
qed

```

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

```

lemma net-asset-value-alt-def:
  net-asset-value α = (∑ i ≤ shortest-period α. π α i)
proof -
  let ?support = {i . π α i ≠ 0}
  {
    fix k
    have (∑ i | i ≤ k ∧ π α i ≠ 0 . π α i) = (∑ i ≤ k. π α i)
    proof (induct k)
      case 0
      thus ?case
      proof (cases π α 0 = 0)
        case True
        then show ?thesis
          by fastforce
      next
        case False
        {
          fix i
          have (i ≤ 0 ∧ π α i ≠ 0) = (i ≤ 0)

```

```

    using False
    by blast
  }
  hence  $(\sum i \mid i \leq 0 \wedge \pi \alpha i \neq 0. \pi \alpha i) =$ 
     $(\sum i \mid i \leq 0. \pi \alpha i)$ 
    by presburger
  also have ... =  $(\sum i \leq 0. \pi \alpha i)$ 
    by simp
  ultimately show ?thesis
    by simp
qed
next
case (Suc k)
then show ?case
proof (cases  $\pi \alpha (Suc\ k) = 0$ )
  case True
  {
    fix i
    have  $(i \leq Suc\ k \wedge \pi \alpha i \neq 0) =$ 
       $(i \leq k \wedge \pi \alpha i \neq 0)$ 
      using True le-Suc-eq by blast
  }
  hence  $(\sum i \mid i \leq Suc\ k \wedge \pi \alpha i \neq 0. \pi \alpha i) =$ 
     $(\sum i \mid i \leq k \wedge \pi \alpha i \neq 0. \pi \alpha i)$ 
    by presburger
  also have ... =  $(\sum i \leq k. \pi \alpha i)$ 
    using Suc by blast
  ultimately show ?thesis using True
    by simp
next
let ?A =  $\{i . i \leq Suc\ k \wedge \pi \alpha i \neq 0\}$ 
let ?A' =  $\{i . i \leq k \wedge \pi \alpha i \neq 0\}$ 
case False
  hence ?A =  $\{i . (i \leq k \wedge \pi \alpha i \neq 0) \vee i = Suc\ k\}$ 
    by auto
  hence ?A = ?A'  $\cup \{i . i = Suc\ k\}$ 
    by (simp add: Collect-disj-eq)
  hence  $\star$ : ?A = ?A'  $\cup \{Suc\ k\}$ 
    by simp
  hence  $\heartsuit$ : finite (?A'  $\cup \{Suc\ k\}$ )
    using finite-nat-set-iff-bounded-le
    by blast
  hence
     $(\sum i \mid i \leq Suc\ k \wedge \pi \alpha i \neq 0. \pi \alpha i) =$ 
       $(\sum i \in ?A' \cup \{Suc\ k\}. \pi \alpha i)$ 
    unfolding  $\star$ 
    by auto
  also have ... =  $(\sum i \in ?A'. \pi \alpha i) + (\sum i \in \{Suc\ k\}. \pi \alpha i)$ 
    using  $\heartsuit$ 

```

```

    by force
  also have ... = (∑ i ∈ ?A'. π α i) + π α (Suc k)
    by simp
  ultimately show ?thesis
    by (simp add: Suc)
qed
qed
}
hence †:
  (∑ i | i ≤ shortest-period α ∧ π α i ≠ 0. π α i) =
    (∑ i ≤ shortest-period α. π α i)
  by auto
{
  fix i
  have (i ≤ shortest-period α ∧ π α i ≠ 0) = (π α i ≠ 0)
    using shortest-period-bound by blast
}
note · = this
show ?thesis
  using †
  unfolding · net-asset-value-def
  by blast
qed

```

```

lemma greater-than-shortest-period-zero:
  assumes shortest-period α < m
  shows π α m = 0
proof -
  let ?support = {i . π α i ≠ 0}
  have ∀ i ∈ ?support . i ≤ shortest-period α
    by (simp add: finite-account-support shortest-period-def)
  then show ?thesis
    using assms
    by (meson CollectI leD)
qed

```

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 α ≤ n
  shows net-asset-value α = (∑ i ≤ n. π α i)
proof (cases shortest-period α = n)
  case True
  then show ?thesis
    unfolding net-asset-value-alt-def by auto
next
  case False
  hence shortest-period α < n using assms by auto

```

hence $(\sum_{i=\text{shortest-period } \alpha + 1.. n. \pi \alpha} i) = 0$
(is $?\Sigma\text{extra} = 0$)
using *greater-than-shortest-period-zero*
by *auto*
moreover have $(\sum_{i \leq n. \pi \alpha} i) =$
 $(\sum_{i \leq \text{shortest-period } \alpha. \pi \alpha} i) + ?\Sigma\text{extra}$
(is $?lhs = ?\Sigma\text{shortest-period} + -$)
by (*metis*
 $\langle \text{shortest-period } \alpha < n \rangle$
Suc-eq-plus1
less-imp-add-positive
sum-up-index-split)
ultimately have $?lhs = ?\Sigma\text{shortest-period}$
by *linarith*
then show $?thesis$
unfolding *net-asset-value-alt-def* **by** *auto*
qed

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 $0 \leq \text{net-asset-value } \alpha$
using *assms strictly-solvent-def net-asset-value-alt-def* **by** *auto*

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*]: $\text{net-asset-value } 0 = 0$
unfolding *net-asset-value-alt-def*
using *zero-account-def* **by** *force*

lemma *net-asset-value-mono*:
assumes $\alpha \leq \beta$
shows $\text{net-asset-value } \alpha \leq \text{net-asset-value } \beta$
proof –
let $?r = \max(\text{shortest-period } \alpha) (\text{shortest-period } \beta)$
have $\text{shortest-period } \alpha \leq ?r$ **and** $\text{shortest-period } \beta \leq ?r$ **by** *auto*
hence $\text{net-asset-value } \alpha = (\sum_{i \leq ?r. \pi \alpha} i)$
and $\text{net-asset-value } \beta = (\sum_{i \leq ?r. \pi \beta} i)$
using *net-asset-value-shortest-period-ge* **by** *presburger+*
thus $?thesis$ **using** *assms unfolding less-eq-account-def* **by** *auto*
qed

lemma *net-asset-value-uminus*: $net\text{-asset}\text{-value} (-\alpha) = -\text{net}\text{-asset}\text{-value} \alpha$
unfolding
net-asset-value-alt-def
shortest-period-uminus
Rep-account-uminus
by (*simp add: sum-negf*)

lemma *net-asset-value-plus*:
 $net\text{-asset}\text{-value} (\alpha + \beta) = net\text{-asset}\text{-value} \alpha + net\text{-asset}\text{-value} \beta$
(is *?lhs = ?Σα + ?Σβ*)

proof –
let *?r = max (shortest-period α) (shortest-period β)*
have *A: shortest-period (α + β) ≤ ?r*
and *B: shortest-period α ≤ ?r*
and *C: shortest-period β ≤ ?r*
using *shortest-period-plus* **by** *presburger+*
have *?lhs = (Σ i ≤ ?r. π (α + β) i)*
using *net-asset-value-shortest-period-ge* [*OF A*] .
also have *... = (Σ i ≤ ?r. π α i + π β i)*
using *Rep-account-plus* **by** *presburger*
ultimately show *?thesis*
using
net-asset-value-shortest-period-ge [*OF B*]
net-asset-value-shortest-period-ge [*OF C*]
by (*simp add: sum.distrib*)
qed

lemma *net-asset-value-minus*:
 $net\text{-asset}\text{-value} (\alpha - \beta) = net\text{-asset}\text{-value} \alpha - net\text{-asset}\text{-value} \beta$
using *additive.diff additive.intro net-asset-value-plus* **by** *blast*

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

lemma *net-asset-value-just-cash-left-inverse*:
 $net\text{-asset}\text{-value} (just\text{-cash} c) = c$
using *net-asset-value-alt-def partial-nav-just-cash* **by** *presburger*

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 ⇒ real ⇒ real*
where *total-interest* $\mathcal{L} i = (\sum a \in UNIV. i * net\text{-asset}\text{-value} (\mathcal{L} a))$

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

proof (*cases a = b*)
case *True*

```

show ?thesis
  unfolding ⟨a = b⟩ transfer-collapse ..
next
case False
have total-interest ?L' i = (∑ a ∈ UNIV . i * net-asset-value (?L' a))
  unfolding total-interest-def ..
also have ... = (∑ a ∈ UNIV - {a, b} ∪ {a,b}. i * net-asset-value (?L' a))
  by (metis Un-Diff-cancel2 Un-UNIV-left)
also have ... = (∑ a ∈ UNIV - {a, b}. i * net-asset-value (?L' a)) +
  i * net-asset-value (?L' a) + i * net-asset-value (?L' b)
  (is - = ?Σ + - + -)
  using ⟨a ≠ b⟩
  by simp
also have ... = ?Σ +
  i * net-asset-value (L a - τ) +
  i * net-asset-value (L b + τ)
  unfolding transfer-def
  using ⟨a ≠ b⟩
  by auto
also have ... = ?Σ +
  i * net-asset-value (L a) +
  i * net-asset-value (- τ) +
  i * net-asset-value (L b) +
  i * net-asset-value τ
  unfolding minus-account-def net-asset-value-plus
  by (simp add: distrib-left)
also have ... = ?Σ +
  i * net-asset-value (L a) +
  i * net-asset-value (L b)
  unfolding net-asset-value-uminus
  by linarith
also have ... = (∑ a ∈ UNIV - {a, b}. i * net-asset-value (L a)) +
  i * net-asset-value (L a) +
  i * net-asset-value (L b)
  unfolding transfer-def
  using ⟨a ≠ b⟩
  by force
also have ... = (∑ a ∈ UNIV - {a, b} ∪ {a,b}. i * net-asset-value (L a))
  using ⟨a ≠ b⟩ by force
ultimately show ?thesis
  unfolding total-interest-def
  by (metis Diff-partition Un-commute top-greatest)
qed

```

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 $\varrho :: \text{nat} \Rightarrow \text{real}$. 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* :: $(\text{nat} \Rightarrow \text{real}) \Rightarrow \text{account} \Rightarrow \text{account}$ **where**
return-loans $\varrho \alpha = \iota (\lambda n . (1 - \varrho n) * \pi \alpha n)$

lemma *Rep-account-return-loans* [*simp*]:
 $\pi (\text{return-loans } \varrho \alpha) = (\lambda n . (1 - \varrho n) * \pi \alpha n)$

proof –

have $(\text{support } 0 \text{ UNIV } (\lambda n . (1 - \varrho n) * \pi \alpha n)) \subseteq$
 $(\text{support } 0 \text{ UNIV } (\pi \alpha))$
unfolding *support-def*
by (*simp add: Collect-mono*)
moreover have *finite* $(\text{support } 0 \text{ UNIV } (\pi \alpha))$
using *Rep-account*
unfolding *fin-support-def* **by** *auto*
ultimately have *finite* $(\text{support } 0 \text{ UNIV } (\lambda n . (1 - \varrho n) * \pi \alpha n))$
using *infinite-super* **by** *blast*
hence $(\lambda n . (1 - \varrho n) * \pi \alpha n) \in \text{fin-support } 0 \text{ UNIV}$
unfolding *fin-support-def* **by** *auto*
thus *?thesis*
using
Rep-account
Abs-account-inject
Rep-account-inverse
return-loans-def
by *auto*
qed

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

definition *update-account* :: $(\text{nat} \Rightarrow \text{real}) \Rightarrow \text{real} \Rightarrow \text{account} \Rightarrow \text{account}$ **where**

$update-account \ \varrho \ i \ \alpha = just-cash \ (i * net-asset-value \ \alpha) + return-loans \ \varrho \ \alpha$

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

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 –

```
{
  fix  $\alpha :: account$ 
  fix  $S :: account \ set$ 
  assume finite  $S$  and  $\alpha \in S$ 
  hence  $shortest-period \ \alpha \leq Max \ (shortest-period \ ' S)$ 
  proof (induct  $S$  rule: finite-induct)
    case empty
    then show ?case by auto
  next
  case (insert  $\beta \ S$ )
  then show ?case
  proof (cases  $\alpha = \beta$ )
    case True
    then show ?thesis
      by (simp add: insert.hyps(1))
  next
  case False
  hence  $\alpha \in S$ 
  using insert.prem1 by fastforce
  then show ?thesis
  by (meson
      Max-ge
      finite-imageI
      finite-insert
      imageI
      insert.hyps(1)
      insert.prem1)
```

```

    qed
  qed
}
moreover
have finite (range  $\mathcal{L}$ )
  by force
ultimately show ?thesis
  by (simp add: shortest-ledger-period-def)
qed

theorem (in finite) update-balanced:
  assumes  $\varrho \ 0 = 0$  and  $\forall n. \varrho \ n < 1$  and  $0 \leq i$ 
  shows balanced  $\mathcal{L} \ c = \text{balanced} \ (\text{update-ledger } \varrho \ i \ \mathcal{L}) \ ((1 + i) * c)$ 
  (is - = balanced ?L' ((1 + i) * c))
proof
  assume balanced  $\mathcal{L} \ c$ 
  have  $\forall n > 0. (\sum_{a \in UNIV}. \pi \ (?L' \ a) \ n) = 0$ 
  proof (rule allI, rule impI)
    fix  $n :: nat$ 
    assume  $n > 0$ 
    {
      fix  $a$ 
      let  $?a' = \lambda n. (1 - \varrho \ n) * \pi \ (\mathcal{L} \ a) \ n$ 
      have  $\pi \ (?L' \ a) \ n = ?a' \ n$ 
      unfolding
        update-ledger-def
        update-account-def
        Rep-account-plus
        Rep-account-just-cash
        Rep-account-return-loans
      using plus-account-def  $\langle n > 0 \rangle$ 
      by simp
    }
  hence  $(\sum_{a \in UNIV}. \pi \ (?L' \ a) \ n) =$ 
     $(1 - \varrho \ n) * (\sum_{a \in UNIV}. \pi \ (\mathcal{L} \ a) \ n)$ 
  using finite-UNIV
  by (metis (mono-tags, lifting) sum.cong sum-distrib-left)
  thus  $(\sum_{a \in UNIV}. \pi \ (?L' \ a) \ n) = 0$ 
  using  $\langle 0 < n \rangle$  balanced  $\mathcal{L} \ c$  local.balanced-alt-def by force
qed
moreover
{
  fix  $S :: 'a \ \text{set}$ 
  let  $?w = \text{shortest-ledger-period } \mathcal{L}$ 
  assume  $(\sum_{a \in S}. \text{cash-reserve} \ (\mathcal{L} \ a)) = c$ 
  and  $\forall n > 0. (\sum_{a \in S}. \pi \ (\mathcal{L} \ a) \ n) = 0$ 
  have  $(\sum_{a \in S}. \text{cash-reserve} \ (?L' \ a)) =$ 
     $(\sum_{a \in S}. i * (\sum_{n \leq ?w}. \pi \ (\mathcal{L} \ a) \ n) +$ 
       $\text{cash-reserve} \ (\mathcal{L} \ a))$ 

```

```

using finite
proof (induct S arbitrary: c rule: finite-induct)
  case empty
  then show ?case
    by auto
next
case (insert x S)
have  $(\sum a \in \text{insert } x \text{ } S. \text{cash-reserve } (?L' a)) =$ 
 $(\sum a \in \text{insert } x \text{ } S. i * (\sum n \leq ?\omega. \pi (\mathcal{L} a) n) +$ 
 $\text{cash-reserve } (\mathcal{L} a))$ 
  unfolding update-ledger-def update-account-def cash-reserve-def
  by (simp add: ‹ $0 = 0$ ›,
    metis (no-types)
      shortest-ledger-period-bound
      net-asset-value-shortest-period-ge)
  thus ?case .
qed
also have ... =  $(\sum a \in S. i * (\sum n = 1 .. ?\omega. \pi (\mathcal{L} a) n) +$ 
 $i * \text{cash-reserve } (\mathcal{L} a) + \text{cash-reserve } (\mathcal{L} a))$ 
  unfolding cash-reserve-def
  by (simp add:
    add.commute
    distrib-left
    sum.atMost-shift
    sum-bounds-lt-plus1)
also have ... =  $(\sum a \in S. i * (\sum n = 1 .. ?\omega. \pi (\mathcal{L} a) n) +$ 
 $(1 + i) * \text{cash-reserve } (\mathcal{L} a))$ 
  using finite
  by (induct S rule: finite-induct, auto, simp add: distrib-right)
also have ... =  $i * (\sum a \in S. (\sum n = 1 .. ?\omega. \pi (\mathcal{L} a) n)) +$ 
 $(1 + i) * (\sum a \in S. \text{cash-reserve } (\mathcal{L} a))$ 
  by (simp add: sum.distrib sum-distrib-left)
also have ... =  $i * (\sum n = 1 .. ?\omega. (\sum a \in S. \pi (\mathcal{L} a) n)) +$ 
 $(1 + i) * c$ 
  using ‹ $(\sum a \in S. \text{cash-reserve } (\mathcal{L} a)) = c$ › sum.swap by force
finally have  $(\sum a \in S. \text{cash-reserve } (?L' a)) = c * (1 + i)$ 
  using ‹ $\forall n > 0. (\sum a \in S. \pi (\mathcal{L} a) n) = 0$ ›
  by simp
}
hence  $(\sum a \in UNIV. \text{cash-reserve } (?L' a)) = c * (1 + i)$ 
  using ‹balanced  $\mathcal{L} c$ ›
  unfolding balanced-alt-def
  by fastforce
ultimately show balanced ?L' ((1 + i) * c)
  unfolding balanced-alt-def
  by auto
next
assume balanced ?L' ((1 + i) * c)
have *:  $\forall n > 0. (\sum a \in UNIV. \pi (\mathcal{L} a) n) = 0$ 

```

proof (*rule allI, rule impI*)
fix $n :: \text{nat}$
assume $n > 0$
hence $0 = (\sum a \in \text{UNIV}. \pi (?L' a) n)$
using $\langle \text{balanced } ?L' ((1 + i) * c) \rangle$
unfolding *balanced-alt-def*
by *auto*
also have $\dots = (\sum a \in \text{UNIV}. (1 - \rho n) * \pi (\mathcal{L} a) n)$
unfolding
update-ledger-def
update-account-def
Rep-account-return-loans
Rep-account-just-cash
using $\langle n > 0 \rangle$
by *auto*
also have $\dots = (1 - \rho n) * (\sum a \in \text{UNIV}. \pi (\mathcal{L} a) n)$
by (*simp add: sum-distrib-left*)
finally show $(\sum a \in \text{UNIV}. \pi (\mathcal{L} a) n) = 0$
by (*metis*
 $\langle \forall r. \rho r < 1 \rangle$
diff-gt-0-iff-gt
less-numeral-extra(3)
mult-eq-0-iff)

qed
moreover
{
fix $S :: 'a \text{ set}$
let $?w = \text{shortest-ledger-period } \mathcal{L}$
assume $(\sum a \in S. \text{cash-reserve } (?L' a)) = (1 + i) * c$
and $\forall n > 0. (\sum a \in S. \pi (\mathcal{L} a) n) = 0$
hence $(1 + i) * c = (\sum a \in S. \text{cash-reserve } (?L' a))$
by *auto*
also have $\dots = (\sum a \in S. i * (\sum n \leq ?w. \pi (\mathcal{L} a) n) + \text{cash-reserve } (\mathcal{L} a))$
using *finite*
proof (*induct S rule: finite-induct*)
case *empty*
then show *?case*
by *auto*
next
case (*insert x S*)
have $(\sum a \in \text{insert } x \text{ } S. \text{cash-reserve } (?L' a)) =$
 $(\sum a \in \text{insert } x \text{ } S.$
 $i * (\sum n \leq ?w. \pi (\mathcal{L} a) n) + \text{cash-reserve } (\mathcal{L} a))$
unfolding *update-ledger-def update-account-def cash-reserve-def*
by (*simp add: \langle \rho 0 = 0 \rangle,*
metis (no-types)
shortest-ledger-period-bound
net-asset-value-shortest-period-ge)

```

thus ?case .
qed
also have ... = (∑ a∈S. i * (∑ n = 1 .. ?ω. π (ℒ a) n) +
                  i * cash-reserve (ℒ a) + cash-reserve (ℒ a))
unfolding cash-reserve-def
by (simp add:
      add.commute
      distrib-left
      sum.atMost-shift
      sum-bounds-lt-plus1)
also have ... = (∑ a∈S. i * (∑ n = 1 .. ?ω. π (ℒ a) n) +
                  (1 + i) * cash-reserve (ℒ a))
using finite
by (induct S rule: finite-induct, auto, simp add: distrib-right)
also have ... = i * (∑ a∈S. (∑ n = 1 .. ?ω. π (ℒ a) n)) +
                  (1 + i) * (∑ a∈S. cash-reserve (ℒ a))
by (simp add: sum.distrib sum-distrib-left)
also have ... = i * (∑ n = 1 .. ?ω. (∑ a∈S. π (ℒ a) n)) +
                  (1 + i) * (∑ a∈S. cash-reserve (ℒ a))
using sum.swap by force
also have ... = (1 + i) * (∑ a∈S. cash-reserve (ℒ a))
using ⟨∀ n>0. (∑ a∈S. π (ℒ a) n) = 0⟩
by simp
finally have c = (∑ a∈S. cash-reserve (ℒ a))
using ⟨0 ≤ i⟩
by force
}
hence (∑ a∈UNIV. cash-reserve (ℒ a)) = c
unfolding cash-reserve-def
by (metis
      Rep-account-just-cash
      ⟨balanced ?ℒ' ((1 + i) * c)⟩
      *
      balanced-def
      finite-Rep-account-ledger)
ultimately show balanced ℒ c
unfolding balanced-alt-def
by auto
qed

```

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 $(\alpha + \beta) = \text{return-loans } \varrho \alpha + \text{return-loans } \varrho \beta$

proof –

let $? \alpha = \pi \alpha$

let $? \beta = \pi \beta$

let $? \varrho \alpha \beta = \lambda n. (1 - \varrho n) * (? \alpha n + ? \beta n)$

let $? \varrho \alpha = \lambda n. (1 - \varrho n) * ? \alpha n$

let $? \varrho \beta = \lambda n. (1 - \varrho n) * ? \beta n$

have *support 0 UNIV* $? \varrho \alpha \subseteq \text{support 0 UNIV } ? \alpha$

support 0 UNIV $? \varrho \beta \subseteq \text{support 0 UNIV } ? \beta$

support 0 UNIV $? \varrho \alpha \beta \subseteq \text{support 0 UNIV } ? \alpha \cup \text{support 0 UNIV } ? \beta$

unfolding *support-def*

by *auto*

moreover have

$? \alpha \in \text{fin-support 0 UNIV}$

$? \beta \in \text{fin-support 0 UNIV}$

using *Rep-account* **by** *force+*

ultimately have \star :

$? \varrho \alpha \in \text{fin-support 0 UNIV}$

$? \varrho \beta \in \text{fin-support 0 UNIV}$

$? \varrho \alpha \beta \in \text{fin-support 0 UNIV}$

unfolding *fin-support-def*

using *finite-subset* **by** *auto+*

{

fix n

have $\pi (\text{return-loans } \varrho (\alpha + \beta)) n =$

$\pi (\text{return-loans } \varrho \alpha + \text{return-loans } \varrho \beta) n$

unfolding *return-loans-def* *Rep-account-plus*

using \star *Abs-account-inverse* *distrib-left* **by** *auto*

}

hence $\pi (\text{return-loans } \varrho (\alpha + \beta)) =$

$\pi (\text{return-loans } \varrho \alpha + \text{return-loans } \varrho \beta)$

by *auto*

thus *?thesis*

by (*metis* *Rep-account-inverse*)

qed

lemma *return-loans-zero* [*simp*]: *return-loans* $\varrho 0 = 0$

proof –

have $(\lambda n. (1 - \varrho n) * 0) = (\lambda -. 0)$

by *force*

hence $\iota (\lambda n. (1 - \varrho n) * 0) = 0$

unfolding *zero-account-def*

by *presburger*

thus *?thesis*

unfolding *return-loans-def* *Rep-account-zero* .

qed

lemma *return-loans-uminus*: $\text{return-loans } \varrho (-\alpha) = - \text{return-loans } \varrho \alpha$
by (*metis*
add.left-cancel
diff-self
minus-account-def
return-loans-plus
return-loans-zero)

lemma *return-loans-subtract*:
 $\text{return-loans } \varrho (\alpha - \beta) = \text{return-loans } \varrho \alpha - \text{return-loans } \varrho \beta$
by (*simp add: additive.diff additive-def return-loans-plus*)

As presented in §1, each index corresponds to a progressively shorter loan period. This is captured by a monotonicity assumption on the rate function $\varrho :: \text{nat} \Rightarrow \text{real}$. 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 $\varrho :: \text{nat} \Rightarrow \text{real}$, we may in turn prove monotonicity for *return-loans* over $(\leq) :: \text{account} \Rightarrow \text{account} \Rightarrow \text{bool}$.

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 $\text{return-loans } \varrho \alpha \leq \text{return-loans } \varrho \beta$

proof –

```
{
  fix  $\alpha :: \text{account}$ 
  assume  $0 \leq \alpha$ 
  {
    fix  $n :: \text{nat}$ 
    let  $?\alpha = \pi \alpha$ 
    let  $? \varrho \alpha = \lambda n. (1 - \varrho n) * ?\alpha n$ 
    have  $\forall n. 0 \leq (\sum_{i \leq n} ?\alpha i)$ 
      using  $\langle 0 \leq \alpha \rangle$ 
      unfolding less-eq-account-def Rep-account-zero
      by simp
    hence  $0 \leq (\sum_{i \leq n} ?\alpha i)$  by auto
    moreover have  $(1 - \varrho n) * (\sum_{i \leq n} ?\alpha i) \leq (\sum_{i \leq n} ? \varrho \alpha i)$ 
    proof (induct n)
      case 0
      then show ?case by simp
    next
      case (Suc n)
      have  $0 \leq (1 - \varrho (\text{Suc } n))$ 
        by (simp add:  $\langle \forall n. \varrho n < 1 \rangle$  less-eq-real-def)
```

moreover have $(1 - \varrho (\text{Suc } n)) \leq (1 - \varrho n)$
using $\langle \forall n m . n \leq m \longrightarrow \varrho n \leq \varrho m \rangle$
by *simp*
ultimately have
 $(1 - \varrho (\text{Suc } n)) * (\sum i \leq n . ?\alpha i) \leq (1 - \varrho n) * (\sum i \leq n . ?\alpha i)$
using $\langle \forall n . 0 \leq (\sum i \leq n . ?\alpha i) \rangle$
by (*meson le-less mult-mono*)
hence
 $(1 - \varrho (\text{Suc } n)) * (\sum i \leq \text{Suc } n . ?\alpha i) \leq$
 $(1 - \varrho n) * (\sum i \leq n . ?\alpha i) + (1 - \varrho (\text{Suc } n)) * (? \alpha (\text{Suc } n))$
(is - \leq ?X)
by (*simp add: distrib-left*)
moreover have
 $?X \leq (\sum i \leq \text{Suc } n . ?\varrho \alpha i)$
using *Suc.hyps* **by** *fastforce*
ultimately show *?case* **by** *auto*
qed
moreover have $0 < 1 - \varrho n$
by (*simp add: $\langle \forall n . \varrho n < 1 \rangle$*)
ultimately have $0 \leq (\sum i \leq n . ?\varrho \alpha i)$
using *dual-order.trans* **by** *fastforce*
}
hence *strictly-solvent (return-loans $\varrho \alpha$)*
unfolding *strictly-solvent-def Rep-account-return-loans*
by *auto*
}
hence $0 \leq \text{return-loans } \varrho (\beta - \alpha)$
using $\langle \alpha \leq \beta \rangle$
by (*simp add: strictly-solvent-alt-def*)
thus *?thesis*
by (*metis*
add-diff-cancel-left'
diff-ge-0-iff-ge
minus-account-def
return-loans-plus)
qed

lemma *return-loans-just-cash:*
assumes $\varrho 0 = 0$
shows $\text{return-loans } \varrho (\text{just-cash } c) = \text{just-cash } c$
proof –
have $(\lambda n. (1 - \varrho n) * \pi (\iota (\lambda n. \text{if } n = 0 \text{ then } c \text{ else } 0))) n$
 $= (\lambda n. \text{if } n = 0 \text{ then } (1 - \varrho n) * c \text{ else } 0)$
using *Rep-account-just-cash just-cash-def* **by** *force*
also have $\dots = (\lambda n. \text{if } n = 0 \text{ then } c \text{ else } 0)$
using $\langle \varrho 0 = 0 \rangle$
by *force*
finally show *?thesis*
unfolding *return-loans-def just-cash-def*

by *presburger*
qed

lemma *distribute-interest-plus*:
 $just-cash (i * net-asset-value (\alpha + \beta)) =$
 $just-cash (i * net-asset-value \alpha) +$
 $just-cash (i * net-asset-value \beta)$
unfolding *just-cash-def net-asset-value-plus*
by (*metis*
distrib-left
just-cash-plus
just-cash-def)

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*:
 $update-account \varrho i (\alpha + \beta) =$
 $update-account \varrho i \alpha + update-account \varrho i \beta$
unfolding
update-account-def
return-loans-plus
distribute-interest-plus
by *simp*

lemma *update-account-zero* [*simp*]: $update-account \varrho i 0 = 0$
by (*metis add-cancel-right-left update-account-plus*)

lemma *update-account-uminus*:
 $update-account \varrho i (-\alpha) = - update-account \varrho i \alpha$
unfolding *update-account-def*
by (*simp add: net-asset-value-uminus return-loans-uminus*)

lemma *update-account-subtract*:
 $update-account \varrho i (\alpha - \beta) =$
 $update-account \varrho i \alpha - update-account \varrho i \beta$
by (*simp add: additive.diff additive.intro update-account-plus*)

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 $update-account \varrho i \alpha \leq update-account \varrho i \beta$
proof –
have $net-asset-value \alpha \leq net-asset-value \beta$
using $\langle \alpha \leq \beta \rangle net-asset-value-mono$ **by** *presburger*
hence $i * net-asset-value \alpha \leq i * net-asset-value \beta$
by (*simp add: $\langle 0 \leq i \rangle mult-left-mono$*)
hence $just-cash (i * net-asset-value \alpha) \leq$

```

      just-cash (i * net-asset-value β)
    by (simp add: just-cash-order-embed)
  moreover
  have return-loans ρ α ≤ return-loans ρ β
    using assms return-loans-mono by presburger
  ultimately show ?thesis unfolding update-account-def
    by (simp add: add-mono)
qed

```

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

```

lemma update-preserves-strictly-solvent:
  assumes 0 ≤ i
  and ∀ n . ρ n < 1
  and ∀ n m . n ≤ m ⟶ ρ n ≤ ρ m
  and strictly-solvent α
  shows strictly-solvent (update-account ρ i α)
  using assms
  unfolding strictly-solvent-alt-def
  by (metis update-account-mono update-account-zero)

```

7 Bulk Update

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

```

primrec bulk-update-account ::
  nat ⇒ (nat ⇒ real) ⇒ real ⇒ account ⇒ account
  where
    bulk-update-account 0 - - α = α
  | bulk-update-account (Suc n) ρ i α =
    update-account ρ i (bulk-update-account n ρ i α)

```

As with $update\text{-}account$, $bulk\text{-}update\text{-}account$ is an order-preserving group homomorphism.

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

```

lemma bulk-update-account-plus:
  bulk-update-account n ρ i (α + β) =
    bulk-update-account n ρ i α + bulk-update-account n ρ i β
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case
    using bulk-update-account.simps(2) update-account-plus by presburger

```

qed

lemma *bulk-update-account-zero* [*simp*]: $\text{bulk-update-account } n \ \varrho \ i \ 0 = 0$
by (*metis add-cancel-right-left bulk-update-account-plus*)

lemma *bulk-update-account-uminus*:
 $\text{bulk-update-account } n \ \varrho \ i \ (-\alpha) = - \text{bulk-update-account } n \ \varrho \ i \ \alpha$
by (*metis add-eq-0-iff bulk-update-account-plus bulk-update-account-zero*)

lemma *bulk-update-account-subtract*:
 $\text{bulk-update-account } n \ \varrho \ i \ (\alpha - \beta) =$
 $\text{bulk-update-account } n \ \varrho \ i \ \alpha - \text{bulk-update-account } n \ \varrho \ i \ \beta$
by (*simp add: additive.diff additive-def bulk-update-account-plus*)

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 $\text{bulk-update-account } n \ \varrho \ i \ \alpha \leq \text{bulk-update-account } n \ \varrho \ i \ \beta$
using *assms*
proof (*induct n*)
 case 0
 then show *?case* **by** *simp*
next
 case (*Suc n*)
 then show *?case*
 using *bulk-update-account.simps(2) update-account-mono* **by** *presburger*
qed

It 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 $0 \leq \text{net-asset-value } (\text{bulk-update-account } n \ \varrho \ i \ \alpha)$
using *assms*
by (*metis*
 bulk-update-account-mono
 bulk-update-account-zero
 strictly-solvent-alt-def
 strictly-solvent-net-asset-value)

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 $loan :: nat \Rightarrow real \Rightarrow account \langle \delta \rangle$
where
 $\delta \ n \ x = \iota (\lambda \ m . \text{if } n = m \text{ then } x \text{ else } 0)$

lemma *loan-just-cash*: $\delta \ 0 \ c = \text{just-cash } c$
unfolding *just-cash-def* *loan-def*
by *force*

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

proof –
have $(\lambda \ m . \text{if } n = m \text{ then } x \text{ else } 0) \in \text{fin-support } 0 \ UNIV$
unfolding *fin-support-def* *support-def*
by *force*
thus *?thesis*
unfolding *loan-def*
using *Abs-account-inverse* **by** *blast*
qed

lemma *loan-zero* [*simp*]: $\delta \ n \ 0 = 0$
unfolding *loan-def*
using *zero-account-def* **by** *fastforce*

lemma *shortest-period-loan*:
assumes $c \neq 0$
shows *shortest-period* $(\delta \ n \ c) = n$
using *assms*
unfolding *shortest-period-def* *Rep-account-loan*
by *simp*

lemma *net-asset-value-loan* [*simp*]: $\text{net-asset-value } (\delta \ n \ c) = c$

proof (*cases* $c = 0$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
hence *shortest-period* $(\delta \ n \ c) = n$ **using** *shortest-period-loan* **by** *blast*
then show *?thesis* **unfolding** *net-asset-value-alt-def* **by** *simp*
qed

lemma *return-loans-loan* [*simp*]: $\text{return-loans } \rho (\delta \ n \ c) = \delta \ n ((1 - \rho \ n) * c)$

proof –
have $\text{return-loans } \rho (\delta \ n \ c) =$
 $\iota (\lambda \ na . (\text{if } n = na \text{ then } (1 - \rho \ n) * c \text{ else } 0))$

```

    unfolding return-loans-def
  by (metis Rep-account-loan mult.commute mult-zero-left)
thus ?thesis
  by (simp add: loan-def)
qed

```

```

lemma account-decomposition:
   $\alpha = (\sum i \leq \text{shortest-period } \alpha. \delta i (\pi \alpha i))$ 
proof -
  let ?p = shortest-period  $\alpha$ 
  let ? $\pi\alpha$  =  $\pi \alpha$ 
  let ? $\Sigma\delta$  =  $\sum i \leq ?p. \delta i (? \pi\alpha i)$ 
  {
    fix n m :: nat
    fix f :: nat  $\Rightarrow$  real
    assume n > m
    hence  $\pi (\sum i \leq m. \delta i (f i)) n = 0$ 
      by (induct m, simp+)
  }
note  $\cdot = \text{this}$ 
{
  fix n :: nat
  have  $\pi ? \Sigma\delta n = ? \pi\alpha n$ 
  proof (cases n  $\leq$  ?p)
  case True
  {
    fix n m :: nat
    fix f :: nat  $\Rightarrow$  real
    assume n  $\leq$  m
    hence  $\pi (\sum i \leq m. \delta i (f i)) n = f n$ 
    proof (induct m)
    case 0
    then show ?case by simp
    next
    case (Suc m)
    then show ?case
    proof (cases n = Suc m)
    case True
    then show ?thesis using  $\cdot$  by auto
    next
    case False
    hence n  $\leq$  m
      using Suc.premis le-Suc-eq by blast
    then show ?thesis
      by (simp add: Suc.hyps)
    qed
  }
  qed
}
then show ?thesis using True by auto

```

```

next
  case False
  have  $?\pi\alpha\ n = 0$ 
    unfolding shortest-period-def
    using False shortest-period-bound by blast
    thus  $?thesis$  using False · by auto
  qed
}
thus  $?thesis$ 
  by (metis Rep-account-inject ext)
qed

```

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* $::\ account \Rightarrow nat \Rightarrow real$ **where**
reserves-for-period $\alpha\ n =$

$$\begin{aligned}
& \text{fold} \\
& \quad \min \\
& \quad \left[\left(\sum_{i \leq k} \pi\ \alpha\ i \right) . k \leftarrow [n..<shortest-period\ \alpha+1] \right] \\
& \quad \left(\sum_{i \leq n} \pi\ \alpha\ i \right)
\end{aligned}$$

lemma *nav-reserves-for-period*:

assumes *shortest-period* $\alpha \leq n$

shows *reserves-for-period* $\alpha\ n = net\text{-}asset\text{-}value\ \alpha$

proof *cases*

assume *shortest-period* $\alpha = n$

hence $[n..<shortest-period\ \alpha+1] = [n]$

by *simp*

hence $\left[\left(\sum_{i \leq k} \pi\ \alpha\ i \right) . k \leftarrow [n..<shortest-period\ \alpha+1] \right] =$
 $\left[\left(\sum_{i \leq n} \pi\ \alpha\ i \right) \right]$

by *simp*

then show $?thesis$

unfolding *reserves-for-period-def*

by (*simp add: <shortest-period* $\alpha = n$ *net-asset-value-alt-def*)

next

assume *shortest-period* $\alpha \neq n$

hence *shortest-period* $\alpha < n$

using *assms order-le-imp-less-or-eq* by *blast*

hence $\left[\left(\sum_{i \leq k} \pi\ \alpha\ i \right) . k \leftarrow [n..<shortest-period\ \alpha+1] \right] = []$

by *force*

hence *reserves-for-period* $\alpha\ n = \left(\sum_{i \leq n} \pi\ \alpha\ i \right)$

unfolding *reserves-for-period-def* by *auto*

then show $?thesis$

using *assms net-asset-value-shortest-period-ge* by *presburger*

qed

lemma *reserves-for-period-exists*:

$$\begin{aligned} \exists m \geq n. \text{reserves-for-period } \alpha \ n = & (\sum_{i \leq m} . \pi \ \alpha \ i) \\ \wedge (\forall u \geq n. (\sum_{i \leq m} . \pi \ \alpha \ i) \leq & (\sum_{i \leq u} . \pi \ \alpha \ i)) \end{aligned}$$

proof –

$$\begin{aligned} \{ \\ \text{fix } j \\ \text{have } \exists m \geq n. (\sum_{i \leq m} . \pi \ \alpha \ i) = & \\ \text{fold} \\ \text{min} \\ [(\sum_{i \leq k} . \pi \ \alpha \ i) . k \leftarrow [n..<j]] & \\ (\sum_{i \leq n} . \pi \ \alpha \ i) & \\ \wedge (\forall u \geq n. u < j \longrightarrow (\sum_{i \leq m} . \pi \ \alpha \ i) \leq & (\sum_{i \leq u} . \pi \ \alpha \ i)) \end{aligned}$$

proof (*induct j*)

case 0

then show *?case by auto*

next

case (*Suc j*)

then show *?case*

proof *cases*

assume $j \leq n$

thus *?thesis*

by (*simp, metis dual-order.refl le-less-Suc-eq*)

next

assume $\neg(j \leq n)$

hence $n < j$ **by** *auto*

obtain m **where**

$m \geq n$

$\forall u \geq n. u < j \longrightarrow (\sum_{i \leq m} . \pi \ \alpha \ i) \leq (\sum_{i \leq u} . \pi \ \alpha \ i)$

$(\sum_{i \leq m} . \pi \ \alpha \ i) =$

fold

min

$[(\sum_{i \leq k} . \pi \ \alpha \ i) . k \leftarrow [n..<j]]$

$(\sum_{i \leq n} . \pi \ \alpha \ i)$

using *Suc by blast*

note $\heartsuit = \text{this}$

hence $\dagger: \text{min } (\sum_{i \leq m} . \pi \ \alpha \ i) (\sum_{i \leq j} . \pi \ \alpha \ i) =$

fold

min

$[(\sum_{i \leq k} . \pi \ \alpha \ i) . k \leftarrow [n..<Suc \ j]]$

$(\sum_{i \leq n} . \pi \ \alpha \ i)$

(**is** $- =$ *?fold*)

using $\langle n < j \rangle$ **by** *simp*

show *?thesis*

proof *cases*

assume $(\sum_{i \leq m} . \pi \ \alpha \ i) < (\sum_{i \leq j} . \pi \ \alpha \ i)$

hence

$\forall u \geq n. u < Suc \ j \longrightarrow (\sum_{i \leq m} . \pi \ \alpha \ i) \leq (\sum_{i \leq u} . \pi \ \alpha \ i)$

```

    by (metis
        ♥(2)
        dual-order.order-iff-strict
        less-Suc-eq)
  thus ?thesis
    using † ⟨m ≥ n⟩ by auto
next
assume *: ¬ ((∑ i≤m . π α i) < (∑ i≤j . π α i))
hence
  ∀ u ≥ n. u < j → (∑ i≤j . π α i) ≤ (∑ i≤u . π α i)
  using ♥(2)
  by auto
hence
  ∀ u ≥ n. u < Suc j → (∑ i≤j . π α i) ≤ (∑ i≤u . π α i)
  by (simp add: less-Suc-eq)
also have ?fold = (∑ i≤j . π α i)
  using † * by linarith
ultimately show ?thesis
  by (metis ⟨n < j⟩ less-or-eq-imp-le)
qed
qed
qed
}
from this obtain m where
  m ≥ n
  (∑ i≤m . π α i) = reserves-for-period α n
  ∀ u ≥ n. u < shortest-period α + 1
  → (∑ i≤m . π α i) ≤ (∑ i≤u . π α i)
unfolding reserves-for-period-def
by blast
note ◇ = this
hence (∑ i≤m . π α i) ≤ (∑ i≤shortest-period α . π α i)
  by (metis
      less-add-one
      nav-reserves-for-period
      net-asset-value-alt-def
      nle-le)
hence ∀ u ≥ shortest-period α. (∑ i≤m . π α i) ≤ (∑ i≤u . π α i)
  by (metis
      net-asset-value-alt-def
      net-asset-value-shortest-period-ge)
hence ∀ u ≥ n. (∑ i≤m . π α i) ≤ (∑ i≤u . π α i)
  by (metis ◇(3) Suc-eq-plus1 less-Suc-eq linorder-not-le)
thus ?thesis
  using ◇(1) ◇(2)
  by metis
qed

```

lemma permissible-loan-converse:

assumes *strictly-solvent* $(\alpha - \delta n c)$

shows $c \leq \text{reserves-for-period } \alpha n$

proof –

obtain m **where**

$n \leq m$

reserves-for-period $\alpha n = (\sum_{i \leq m} \pi \alpha i)$

using *reserves-for-period-exists* **by** *blast*

have $(\sum_{i \leq m} \pi (\alpha - \delta n c) i) = (\sum_{i \leq m} \pi \alpha i) - c$

using $\langle n \leq m \rangle$

proof (*induct* m)

case 0

hence $n = 0$ **by** *auto*

have $\pi (\alpha - \delta n c + \delta n c) 0 = \pi (\alpha - \delta n c) 0 + \pi (\delta n c) 0$

using *Rep-account-plus* **by** *presburger*

thus *?case*

unfolding $\langle n = 0 \rangle$

by *simp*

next

case (*Suc* m)

then show *?case*

proof *cases*

assume $n = \text{Suc } m$

hence $m < n$ **by** *auto*

hence $(\sum_{i \leq m} \pi (\alpha - \delta n c) i) = (\sum_{i \leq m} \pi \alpha i)$

proof(*induct* m)

case 0

then show *?case*

by (*metis*

(no-types, opaque-lifting)

Rep-account-loan

Rep-account-plus

atMost-0 bot-nat-0.not-eq-extremum

diff-0-right

diff-add-cancel

empty-iff

finite.intros(1)

sum.empty

sum.insert)

next

case (*Suc* m)

hence $m < n$ **and** $n \neq \text{Suc } m$

using *Suc-lessD* **by** *blast+*

moreover have

$\pi (\alpha - \delta n c + \delta n c) (\text{Suc } m) =$

$\pi (\alpha - \delta n c) (\text{Suc } m) + \pi (\delta n c) (\text{Suc } m)$

using *Rep-account-plus* **by** *presburger*

ultimately show *?case* **by** (*simp add: Suc.hyps*)

qed

moreover

have $\pi (\alpha - \delta (\text{Suc } m) c + \delta (\text{Suc } m) c) (\text{Suc } m) =$
 $\pi (\alpha - \delta (\text{Suc } m) c) (\text{Suc } m) + \pi (\delta (\text{Suc } m) c) (\text{Suc } m)$
by (*meson Rep-account-plus*)
ultimately show ?thesis
unfolding $\langle n = \text{Suc } m \rangle$
by *simp*
next
assume $n \neq \text{Suc } m$
hence $n \leq m$
using *Suc.premis le-SucE* **by** *blast*
have $\pi (\alpha - \delta n c + \delta n c) (\text{Suc } m) =$
 $\pi (\alpha - \delta n c) (\text{Suc } m) + \pi (\delta n c) (\text{Suc } m)$
by (*meson Rep-account-plus*)
moreover have $0 = (\text{if } n = \text{Suc } m \text{ then } c \text{ else } 0)$
using $\langle n \neq \text{Suc } m \rangle$ **by** *presburger*
ultimately show ?thesis
by (*simp add: Suc.hyps* $\langle n \leq m \rangle$)
qed
qed
hence $0 \leq (\sum i \leq m . \pi \alpha i) - c$
by (*metis assms strictly-solvent-def*)
thus ?thesis
by (*simp add: reserves-for-period* $\alpha n = \text{sum } (\pi \alpha) \{..m\}$)
qed

lemma *permissible-loan*:

assumes *strictly-solvent* α

shows *strictly-solvent* $(\alpha - \delta n c) = (c \leq \text{reserves-for-period } \alpha n)$

proof

assume *strictly-solvent* $(\alpha - \delta n c)$

thus $c \leq \text{reserves-for-period } \alpha n$

using *permissible-loan-converse* **by** *blast*

next

assume $c \leq \text{reserves-for-period } \alpha n$

{

fix j

have $0 \leq (\sum i \leq j . \pi (\alpha - \delta n c) i)$

proof *cases*

assume $j < n$

hence $(\sum i \leq j . \pi (\alpha - \delta n c) i) = (\sum i \leq j . \pi \alpha i)$

proof (*induct j*)

case 0

then show ?case

by (*simp*,

metis

Rep-account-loan

Rep-account-plus

$\langle j < n \rangle$

add commute)

add-0
diff-add-cancel
gr-implies-not-zero)

next
case (*Suc j*)
moreover have $\pi (\alpha - \delta n c + \delta n c) (\text{Suc } j) =$
 $\pi (\alpha - \delta n c) (\text{Suc } j) + \pi (\delta n c) (\text{Suc } j)$
using *Rep-account-plus* **by** *presburger*
ultimately show *?case* **by** *simp*
qed
thus *?thesis*
by (*metis* *assms* *strictly-solvent-def*)

next
assume $\neg (j < n)$
hence $n \leq j$ **by** *auto*
obtain *m* **where**
reserves-for-period $\alpha n = (\sum i \leq m . \pi \alpha i)$
 $\forall u \geq n. (\sum i \leq m . \pi \alpha i) \leq (\sum i \leq u . \pi \alpha i)$
using *reserves-for-period-exists* **by** *blast*
hence $\forall u \geq n. c \leq (\sum i \leq u . \pi \alpha i)$
using $\langle c \leq \text{reserves-for-period } \alpha n \rangle$
by *auto*
hence $c \leq (\sum i \leq j . \pi \alpha i)$
using $\langle n \leq j \rangle$ **by** *presburger*
hence $0 \leq (\sum i \leq j . \pi \alpha i) - c$
by *force*
moreover have $(\sum i \leq j . \pi \alpha i) - c = (\sum i \leq j . \pi (\alpha - \delta n c) i)$
using $\langle n \leq j \rangle$
proof (*induct j*)
case *0*
hence $n = 0$ **by** *auto*
have $\pi (\alpha - \delta 0 c + \delta 0 c) 0 = \pi (\alpha - \delta 0 c) 0 + \pi (\delta 0 c) 0$
using *Rep-account-plus* **by** *presburger*
thus *?case* **unfolding** $\langle n = 0 \rangle$ **by** *simp*

next
case (*Suc j*)
then show *?case*
proof *cases*
assume $n = \text{Suc } j$
hence $j < n$
by *blast*
hence $(\sum i \leq j . \pi (\alpha - \delta n c) i) = (\sum i \leq j . \pi \alpha i)$
proof (*induct j*)
case *0*
then show *?case*
by (*simp*,
metis
Rep-account-loan
Rep-account-plus)

```

      ⟨j < n⟩
      add.commute
      add-0
      diff-add-cancel
      gr-implies-not-zero)
next
  case (Suc j)
  moreover have  $\pi (\alpha - \delta n c + \delta n c) (Suc j) =$ 
     $\pi (\alpha - \delta n c) (Suc j) + \pi (\delta n c) (Suc j)$ 
    using Rep-account-plus by presburger
  ultimately show ?case by simp
qed
moreover have
   $\pi (\alpha - \delta (Suc j) c + \delta (Suc j) c) (Suc j) =$ 
   $\pi (\alpha - \delta (Suc j) c) (Suc j) + \pi (\delta (Suc j) c) (Suc j)$ 
  using Rep-account-plus by presburger
ultimately show ?thesis
  unfolding ⟨n = Suc j⟩
  by simp
next
  assume n ≠ Suc j
  hence n ≤ j
    using Suc.premis le-SucE by blast
  hence  $(\sum_{i \leq j} \pi \alpha i) - c = (\sum_{i \leq j} \pi (\alpha - \delta n c) i)$ 
    using Suc.hyps by blast
  moreover have  $\pi (\alpha - \delta n c + \delta n c) (Suc j) =$ 
     $\pi (\alpha - \delta n c) (Suc j) + \pi (\delta n c) (Suc j)$ 
    using Rep-account-plus by presburger
  ultimately show ?thesis
    by (simp add: ⟨n ≠ Suc j⟩)
qed
qed
ultimately show ?thesis by auto
qed
}
thus strictly-solvent (α - δ n c)
  unfolding strictly-solvent-def
  by auto
qed

```

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 $\text{bulk-update-account } n \ \varrho \ i \ (\text{just-cash } c) =$
 $\text{just-cash } ((1 + i) \wedge n * c)$

proof (*induct n*)
case 0
then show ?*case* **by** *simp*
next
case (*Suc n*)
have *return-loans* ϱ (*just-cash* $((1 + i) ^ n * c)$) =
just-cash $((1 + i) ^ n * c)$
using *assms return-loans-just-cash* **by** *blast*
thus ?*case*
using *Suc net-asset-value-just-cash-left-inverse*
by (*simp add: update-account-def*,
metis
add.commute
mult.commute
mult.left-commute
mult-1
ring-class.ring-distrib(2))

qed

lemma *bulk-update-loan-closed-form*:

assumes $\varrho k \neq 1$

and $\varrho k > 0$

and $\varrho 0 = 0$

and $i \geq 0$

shows *bulk-update-account* $n \varrho i (\delta k c)$ =
just-cash $(c * i * ((1 + i) ^ n - (1 - \varrho k) ^ n) / (i + \varrho k))$
 $+ \delta k ((1 - \varrho k) ^ n * c)$

proof (*induct n*)

case 0

then show ?*case*

by (*simp add: zero-account-alt-def*)

next

case (*Suc n*)

have $i + \varrho k > 0$

using *assms(2) assms(4)* **by** *force*

hence $(i + \varrho k) / (i + \varrho k) = 1$

by *force*

hence *bulk-update-account* (*Suc n*) $\varrho i (\delta k c)$ =

just-cash
 $((c * i) / (i + \varrho k) * (1 + i) * ((1 + i) ^ n - (1 - \varrho k) ^ n) +$
 $c * i * (1 - \varrho k) ^ n * ((i + \varrho k) / (i + \varrho k)))$
 $+ \delta k ((1 - \varrho k) ^ (n + 1) * c)$

using *Suc*

by (*simp add:*

return-loans-plus

$\langle \varrho 0 = 0 \rangle$

return-loans-just-cash

update-account-def

net-asset-value-plus)

net-asset-value-just-cash-left-inverse
add.commute
add.left-commute
distrib-left
mult.assoc
add-divide-distrib
distrib-right
mult.commute
mult.left-commute)

also have

$\dots =$
just-cash
 $((c * i) / (i + \varrho k) * (1 + i) * ((1 + i) ^ n - (1 - \varrho k) ^ n) +$
 $(c * i) / (i + \varrho k) * (1 - \varrho k) ^ n * (i + \varrho k))$
 $+ \delta k ((1 - \varrho k) ^ (n + 1) * c)$
by (*metis (no-types, lifting) times-divide-eq-left times-divide-eq-right*)

also have

$\dots =$
just-cash
 $((c * i) / (i + \varrho k) * ($
 $(1 + i) * ((1 + i) ^ n - (1 - \varrho k) ^ n)$
 $+ (1 - \varrho k) ^ n * (i + \varrho k)))$
 $+ \delta k ((1 - \varrho k) ^ (n + 1) * c)$
by (*metis (no-types, lifting) mult.assoc ring-class.ring-distrib(1)*)

also have

$\dots =$
just-cash
 $((c * i) / (i + \varrho k) * ((1 + i) ^ (n + 1) - (1 - \varrho k) ^ (n + 1)))$
 $+ \delta k ((1 - \varrho k) ^ (n + 1) * c)$
by (*simp add: mult.commute mult-diff-mult*)

ultimately show *?case* **by** *simp*

qed

We next give an *algebraic* closed form. This uses the ordered abelian group that *accounts* 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 *bulk-update-account* $n \ \varrho \ i \ \alpha$

$=$ *just-cash* (
 $(1 + i) ^ n * (\text{cash-reserve } \alpha)$
 $+ (\sum k = 1..shortest-period \ \alpha.$
 $(\pi \ \alpha \ k) * i * ((1 + i) ^ n - (1 - \varrho k) ^ n)$
 $/ (i + \varrho k))$
 $)$
 $+ (\sum k = 1..shortest-period \ \alpha. \delta k ((1 - \varrho k) ^ n * \pi \ \alpha \ k))$

proof –

```

{
  fix m
  have  $\forall k \in \{1..m\}. \varrho k \neq 1 \wedge \varrho k > 0$ 
    by (metis
        assms(2)
        assms(3)
        assms(4)
        atLeastAtMost-iff
        dual-order.refl
        less-numeral-extra(1)
        linorder-not-less
        not-gr-zero)
  hence  $\star: \forall k \in \{1..m\}.$ 
    bulk-update-account n  $\varrho$  i ( $\delta k (\pi \alpha k)$ )
    = just-cash  $((\pi \alpha k) * i * ((1 + i) ^ n - (1 - \varrho k) ^ n) / (i + \varrho k))$ 
      +  $\delta k ((1 - \varrho k) ^ n * (\pi \alpha k))$ 
    using assms(1) assms(4) bulk-update-loan-closed-form by blast
  have bulk-update-account n  $\varrho$  i  $(\sum k \leq m. \delta k (\pi \alpha k))$ 
    =  $(\sum k \leq m. \text{bulk-update-account } n \ \varrho \ i \ (\delta k (\pi \alpha k)))$ 
    by (induct m, simp, simp add: bulk-update-account-plus)
  also have
    ... = bulk-update-account n  $\varrho$  i ( $\delta 0 (\pi \alpha 0)$ )
      +  $(\sum k = 1..m. \text{bulk-update-account } n \ \varrho \ i \ (\delta k (\pi \alpha k)))$ 
    by (simp add: atMost-atLeast0 sum.atLeast-Suc-atMost)
  also have
    ... = just-cash  $((1 + i) ^ n * \text{cash-reserve } \alpha)$ 
      +  $(\sum k = 1..m. \text{bulk-update-account } n \ \varrho \ i \ (\delta k (\pi \alpha k)))$ 
    using
       $\langle \varrho 0 = 0 \rangle$ 
      bulk-update-just-cash-closed-form
      loan-just-cash
      cash-reserve-def
    by presburger
  also have
    ... = just-cash  $((1 + i) ^ n * \text{cash-reserve } \alpha)$ 
      +  $(\sum k = 1..m. \text{just-cash } ((\pi \alpha k) * i * ((1 + i) ^ n - (1 - \varrho k) ^ n) / (i + \varrho k))$ 
        +  $\delta k ((1 - \varrho k) ^ n * (\pi \alpha k)))$ 
    using  $\star$  by auto
  also have
    ... = just-cash  $((1 + i) ^ n * \text{cash-reserve } \alpha)$ 
      +  $(\sum k = 1..m. \text{just-cash } ((\pi \alpha k) * i * ((1 + i) ^ n - (1 - \varrho k) ^ n) / (i + \varrho k)))$ 
        +  $(\sum k = 1..m. \delta k ((1 - \varrho k) ^ n * (\pi \alpha k)))$ 
    by (induct m, auto)
  also have

```

$$\dots = \text{just-cash } ((1 + i) \wedge n * \text{cash-reserve } \alpha)$$

$$+ \text{just-cash } (\sum_{k=1..m} (\pi \alpha k) * i * ((1 + i) \wedge n - (1 - \varrho k) \wedge n) / (i + \varrho k))$$

$$+ (\sum_{k=1..m} \delta k ((1 - \varrho k) \wedge n * (\pi \alpha k)))$$
by (*induct m, auto, metis (no-types, lifting) add.assoc just-cash-plus*)
ultimately have

$$\text{bulk-update-account } n \ \varrho \ i \ (\sum_{k \leq m} \delta k (\pi \alpha k)) =$$

$$\text{just-cash } ($$

$$(1 + i) \wedge n * \text{cash-reserve } \alpha$$

$$+ (\sum_{k=1..m} (\pi \alpha k) * i * ((1 + i) \wedge n - (1 - \varrho k) \wedge n) / (i + \varrho k)))$$

$$+ (\sum_{k=1..m} \delta k ((1 - \varrho k) \wedge n * (\pi \alpha k)))$$
by simp
}
note $\cdot = \text{this}$
have

$$\text{bulk-update-account } n \ \varrho \ i \ \alpha$$

$$= \text{bulk-update-account } n \ \varrho \ i \ (\sum_{k \leq \text{shortest-period } \alpha} \delta k (\pi \alpha k))$$
using *account-decomposition* **by** *presburger*
thus *?thesis unfolding* $\cdot \cdot$
qed

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 $\text{bulk-update-account } n \ \varrho \ i \ \alpha$

$= \iota (\lambda k .$

if $k = 0$ *then*

$(1 + i) \wedge n * (\text{cash-reserve } \alpha)$

$+ (\sum_{j=1..shortest-period \ \alpha} (\pi \alpha j) * i * ((1 + i) \wedge n - (1 - \varrho j) \wedge n)$

$/ (i + \varrho j))$

else

$(1 - \varrho k) \wedge n * \pi \alpha k$

)

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

proof –

obtain ν **where** $X: \nu = \ ?\nu$ **by** *blast*

moreover obtain ν' **where** $Y:$

$\nu' = \pi (\text{just-cash } ($

$(1 + i) \wedge n * (\text{cash-reserve } \alpha)$

$+ (\sum_{j=1..shortest-period \ \alpha} (\pi \alpha j) * i * ((1 + i) \wedge n - (1 - \varrho j) \wedge n)$

$(\pi \alpha j) * i * ((1 + i) \wedge n - (1 - \varrho j) \wedge n)$

```

      / (i + ρ j))
    )
  + (∑ j = 1..shortest-period α. δ j ((1 - ρ j) ^ n * π α j))
  by blast
moreover
{
  fix k
  have ∀ k > shortest-period α . ν k = ν' k
  proof (rule allI, rule impI)
    fix k
    assume shortest-period α < k
    hence ν k = 0
      unfolding X
      by (simp add: greater-than-shortest-period-zero)
    moreover have ν' k = 0
  proof -
    have ∀ c. π (just-cash c) k = 0
      using
        Rep-account-just-cash
        ⟨shortest-period α < k⟩
        just-cash-def
      by auto
    moreover
    have ∀ m < k. π (∑ j = 1..m. δ j ((1 - ρ j) ^ n * π α j)) k = 0
  proof (rule allI, rule impI)
    fix m
    assume m < k
    let ?πΣδ = π (∑ j = 1..m. δ j ((1 - ρ j) ^ n * π α j))
    have ?πΣδ k = (∑ j = 1..m. π (δ j ((1 - ρ j) ^ n * π α j))) k
      by (induct m, auto)
    also have ... = (∑ j = 1..m. 0)
      using ⟨m < k⟩
      by (induct m, simp+)
    finally show ?πΣδ k = 0
      by force
  qed
  ultimately show ?thesis unfolding Y
    using ⟨shortest-period α < k⟩ by force
  qed
  ultimately show ν k = ν' k by auto
  qed
moreover have ∀ k . 0 < k → k ≤ shortest-period α → ν k = ν' k
  proof (rule allI, (rule impI)+)
    fix k
    assume 0 < k
    and k ≤ shortest-period α
    have ν k = (1 - ρ k) ^ n * π α k
      unfolding X
      using ⟨0 < k⟩ by fastforce

```

```

moreover have  $\nu' k = (1 - \varrho k) \wedge n * \pi \alpha k$ 
proof –
  have  $\forall c. \pi (\text{just-cash } c) k = 0$ 
    using  $\langle 0 < k \rangle$  by auto
moreover
{
  fix  $m$ 
  assume  $k \leq m$ 
  have  $\pi (\sum j = 1..m. \delta j ((1 - \varrho j) \wedge n * \pi \alpha j)) k$ 
     $= (\sum j = 1..m. \pi (\delta j ((1 - \varrho j) \wedge n * \pi \alpha j)) k)$ 
    by (induct m, auto)
  also
  have  $\dots = (1 - \varrho k) \wedge n * \pi \alpha k$ 
    using  $\langle 0 < k \rangle \langle k \leq m \rangle$ 
  proof (induct m)
    case  $0$ 
    then show ?case by simp
  next
  case (Suc m)
  then show ?case
  proof (cases k = Suc m)
    case True
    hence  $k > m$  by auto
    hence  $(\sum j = 1..m. \pi (\delta j ((1 - \varrho j) \wedge n * \pi \alpha j)) k) = 0$ 
      by (induct m, auto)
    then show ?thesis
      using  $\langle k > m \rangle \langle k = \text{Suc } m \rangle$ 
      by simp
  next
  case False
  hence  $(\sum j = 1..m. \pi (\delta j ((1 - \varrho j) \wedge n * \pi \alpha j)) k)$ 
     $= (1 - \varrho k) \wedge n * \pi \alpha k$ 
    using Suc.hyps Suc.prem1 Suc.prem2 le-Suc-eq by blast
  moreover have  $k \leq m$ 
    using False Suc.prem2 le-Suc-eq by blast
  ultimately show ?thesis using  $\langle 0 < k \rangle$  by simp
  qed
qed
finally have
   $\pi (\sum j = 1..m. \delta j ((1 - \varrho j) \wedge n * \pi \alpha j)) k$ 
     $= (1 - \varrho k) \wedge n * \pi \alpha k .$ 
}
hence
 $\forall m \geq k.$ 
   $\pi (\sum j = 1..m. \delta j ((1 - \varrho j) \wedge n * \pi \alpha j)) k$ 
     $= (1 - \varrho k) \wedge n * \pi \alpha k$  by auto
ultimately show ?thesis
unfolding  $Y$ 
using  $\langle k \leq \text{shortest-period } \alpha \rangle$ 

```

```

    by force
  qed
  ultimately show  $\nu k = \nu' k$ 
    by fastforce
  qed
  moreover have  $\nu 0 = \nu' 0$ 
  proof -
    have  $\nu 0 = (1 + i)^n * (\text{cash-reserve } \alpha)$ 
      +  $(\sum_{j=1..shortest-period} \alpha.$ 
         $(\pi \alpha j) * i * ((1 + i)^n - (1 - \rho j)^n)$ 
         $/ (i + \rho j))$ 
    using X by presburger
  moreover
  have  $\nu' 0 = (1 + i)^n * (\text{cash-reserve } \alpha)$ 
    +  $(\sum_{j=1..shortest-period} \alpha.$ 
       $(\pi \alpha j) * i * ((1 + i)^n - (1 - \rho j)^n)$ 
       $/ (i + \rho j))$ 
  proof -
    {
      fix m
      have  $\pi (\sum_{j=1..m} \delta j ((1 - \rho j)^n * \pi \alpha j)) 0 = 0$ 
        by (induct m, simp+)
    }
    thus ?thesis unfolding Y
      by simp
  qed
  ultimately show ?thesis by auto
  qed
  ultimately have  $\nu k = \nu' k$ 
    by (metis linorder-not-less not-gr0)
}
hence  $\iota \nu = \iota \nu'$ 
  by presburger
ultimately show ?thesis
  using
    Rep-account-inverse
    assms
    bulk-update-algebraic-closed-form
  by presburger
qed
end

```