Actuarial Mathematics

Yosuke Ito

January 25, 2022

Abstract

Actuarial Mathematics is a theory in applied mathematics, which is mainly used for determining the prices of insurance products and evaluating the liability of a company associating with insurance contracts. It is related to calculus, probability theory and financial theory, etc.

In this entry, I formalize the very basic part of Actuarial Mathematics in Isabelle/HOL. The first formalization is about the theory of interest which deals with interest rates, present value factors, an annuity certain, etc.

I have already formalized the basic part of Actuarial Mathematics in Coq (https://github.com/Yosuke-Ito-345/Actuary). This entry is currently the partial translation and a little generalization of the Coq formalization. The further translation in Isabelle/HOL is now proceeding.

Contents

theory *Preliminaries*
  imports *HOL−Analysis.Analysis*
begin

notation *powr* (**infixr** .^ *80*)

1 Preliminary Definitions and Lemmas

lemma *seq-part-multiple*: **fixes** $m\ n :: nat$ **assumes** $m \neq 0$ **defines** $A \equiv \lambda i{::}nat.\ \{i{*}m\ ..{<}\ (i{+}1){*}m\}$
  shows $\forall\, i\ j.\ i \neq j \longrightarrow A\ i \cap A\ j = \{\}$ **and** $(\bigcup i{<}n.\ A\ i) = \{..{<}\ n{*}m\}$
proof −
  { fix $i\ j :: nat$

```
    have i ≠ j ⟹ A i ∩ A j = {}
    proof (erule contrapos-np)
      assume A i ∩ A j ≠ {}
      then obtain k where k ∈ A i ∩ A j by blast
      hence i*m < (j+1)*m ∧ j*m < (i+1)*m unfolding A-def by force
      hence i < j+1 ∧ j < i+1 using mult-less-cancel2 by blast
      thus i = j by force
    qed }
  thus ∀ i j. i ≠ j ⟶ A i ∩ A j = {} by blast
next
  show (⋃ i<n. A i) = {..< n*m}
  proof
    show (⋃ i<n. A i) ⊆ {..< n*m}
    proof
      fix x::nat
      assume x ∈ (⋃ i<n. A i)
     then obtain i where i-n: i < n and i-x: x < (i+1)*m unfolding A-def by
force
      hence i+1 ≤ n by linarith
      hence x < n*m by (meson less-le-trans mult-le-cancel2 i-x)
      thus x ∈ {..< n*m}
        using diff-mult-distrib mult-1 i-n by auto
    qed
  next
    show {..< n*m} ⊆ (⋃ i<n. A i)
    proof
      fix x::nat
      let ?i = x div m
      assume x ∈ {..< n*m}
      hence ?i < n by (simp add: less-mult-imp-div-less)
      moreover have ?i*m ≤ x ∧ x < (?i+1)*m
        using assms div-times-less-eq-dividend dividend-less-div-times by auto
      ultimately show x ∈ (⋃ i<n. A i) unfolding A-def by force
    qed
  qed
qed

lemma(in field) divide-mult-cancel[simp]: fixes a b assumes b ≠ 0
  shows a / b * b = a
  by (simp add: assms)

lemma inverse-powr: (1/a).^b = a.^-b if a > 0 for a b :: real
  by (smt that powr-divide powr-minus-divide powr-one-eq-one)

lemma powr-eq-one-iff-gen[simp]: a.^x = 1 ⟷ x = 0 if a > 0 a ≠ 1 for a x ::
real
  by (metis powr-eq-0-iff powr-inj powr-zero-eq-one that)

lemma powr-less-cancel2: 0 < a ⟹ 0 < x ⟹ 0 < y ⟹ x.^a < y.^a ⟹ x <
```

```
y
  for a x y ::real
proof −
  assume a-pos: 0 < a and x-pos: 0 < x and y-pos: 0 < y
  show x.^a < y.^a ⟹ x < y
  proof (erule contrapos-pp)
    assume ¬ x < y
    hence x ≥ y by fastforce
    hence x.^a ≥ y.^a
    proof (cases x = y)
      case True
      thus ?thesis by simp
    next
      case False
      hence x.^a > y.^a
        using ‹x ≥ y› powr-less-mono2 a-pos y-pos by auto
      thus ?thesis by auto
    qed
    thus ¬ x.^a < y.^a by fastforce
  qed
qed

lemma geometric-increasing-sum-aux: (1−r)^2 * (∑ k<n. (k+1)*r^k) = 1 −
(n+1)*r^n + n*r^(n+1)
  for n::nat and r::real
proof (induct n)
  case 0
  thus ?case by simp
next
  case (Suc n)
  thus ?case
    by (simp add: distrib-left power2-diff field-simps power2-eq-square)
qed

lemma geometric-increasing-sum: (∑ k<n. (k+1)*r^k) = (1 − (n+1)*r^n +
n*r^(n+1)) / (1−r)^2
  if r ≠ 1 for n::nat and r::real
  by (subst geometric-increasing-sum-aux[THEN sym], simp add: that)

lemma Reals-UNIV[simp]: ℝ = {x::real. True}
  unfolding Reals-def by auto

lemma DERIV-fun-powr2:
  fixes a::real
  assumes a-pos: a > 0
    and f: DERIV f x :> r
  shows DERIV (λx. a.^(f x)) x :> a.^(f x) * r * ln a
proof −
  let ?g = (λx. a)
```

```
  have g: DERIV ?g x :> 0 by simp
  have pos: ?g x > 0 by (simp add: a-pos)
  show ?thesis
    using DERIV-powr[OF g pos f] a-pos by (auto simp add: field-simps)
qed

lemma has-real-derivative-powr2:
  assumes a-pos: a > 0
  shows ((λx. a.^x) has-real-derivative a.^x * ln a) (at x)
proof −
  let ?f = (λx. x::real)
  have f: DERIV ?f x :> 1 by simp
  thus ?thesis using DERIV-fun-powr2[OF a-pos f] by simp
qed

lemma has-integral-powr2-from-0:
  fixes a c :: real
  assumes a-pos: a > 0 and a-neq-1: a ≠ 1 and c-nneg: c ≥ 0
  shows ((λx. a.^x) has-integral ((a.^c − 1) / (ln a))) {0..c}
proof −
  have ((λx. a.^x) has-integral ((a.^c)/(ln a) − (a.^0)/(ln a))) {0..c}
  proof (rule fundamental-theorem-of-calculus[OF c-nneg])
    fix x::real
    assume x ∈ {0..c}
    show ((λy. a.^y / ln a) has-vector-derivative a.^x) (at x within {0..c})
      using has-real-derivative-powr2[OF a-pos, of x]
      apply −
      apply (drule DERIV-cdivide[where c = ln a], simp add: assms)
    apply (rule has-vector-derivative-within-subset[where S=UNIV and T={0..c}],
auto)
     by (rule iffD1[OF has-field-derivative-iff-has-vector-derivative])
  qed
  thus ?thesis
    using assms powr-zero-eq-one by (simp add: field-simps)
qed

lemma integrable-on-powr2-from-0:
  fixes a c :: real
  assumes a-pos: a > 0 and a-neq-1: a ≠ 1 and c-nneg: c ≥ 0
  shows (λx. a.^x) integrable-on {0..c}
 using has-integral-powr2-from-0[OF assms] unfolding integrable-on-def by blast

lemma integrable-on-powr2-from-0-general:
  fixes a c :: real
  assumes a-pos: a > 0 and c-nneg: c ≥ 0
  shows (λx. a.^x) integrable-on {0..c}
proof (cases a = 1)
  case True
  thus ?thesis
```

using *has-integral-const-real* **by** *auto*
next
case *False*
thus *?thesis*
using *has-integral-powr2-from-0 False assms* **by** *auto*
qed

lemma *has-integral-null-interval*: **fixes** *a b* :: *real* **and** *f*::*real* ⇒ *real* **assumes** *a* ≥ *b*
shows (*f has-integral 0*) {*a*..*b*}
using *assms content-real-eq-0* **by** *blast*

lemma *has-integral-interval-reverse*: **fixes** *f* :: *real* ⇒ *real* **and** *a b* :: *real*
assumes *a* ≤ *b*
and *continuous-on* {*a*..*b*} *f*
shows ((λ*x*. *f* (*a*+*b*−*x*)) *has-integral* (*integral* {*a*..*b*} *f*)) {*a*..*b*}
proof −
let *?g* = λ*x*. *a* + *b* − *x*
let *?g′* = λ*x*. −*1*
have *g-C0*: *continuous-on* {*a*..*b*} *?g* **using** *continuous-on-op-minus* **by** *simp*
have *Dg-g′*: ⋀*x*. *x*∈{*a*..*b*} ⟹ (*?g has-field-derivative ?g′ x*) (*at x within* {*a*..*b*})
by (*auto intro*!: *derivative-eq-intros*)
show *?thesis*
using *has-integral-substitution-general*
[*of* {} *a b ?g a b f*, *simplified*, *OF assms g-C0 Dg-g′*, *simplified*]
apply (*simp add*: *has-integral-null-interval*[*OF assms*(*1*), *THEN integral-unique*])
by (*simp add*: *has-integral-neg-iff*)
qed

end
theory *Interest*
imports *Preliminaries*
begin

2 List of Actuarial Notations (Global Scope)

definition *i-nom* :: *real* ⇒ *nat* ⇒ *real* (\$*i*[-]⁀{-} [*0*,*0*] *200*)
where \$*i*[*i*]⁀{*m*} ≡ *m* ∗ ((*1*+*i*).⁀(*1*/*m*) − *1*) — nominal interest rate
definition *i-force* :: *real* ⇒ *real* (\$*δ*[-] [*0*] *200*)
where \$*δ*[*i*] ≡ *ln* (*1*+*i*) — force of interest
definition *d-nom* :: *real* ⇒ *nat* ⇒ *real* (\$*d*[-]⁀{-} [*0*,*0*] *200*)
where \$*d*[*i*]⁀{*m*} ≡ \$*i*[*i*]⁀{*m*} / (*1* + \$*i*[*i*]⁀{*m*}/*m*) — discount rate
abbreviation *d-nom-yr* :: *real* ⇒ *real* (\$*d*[-] [*0*] *200*)
where \$*d*[*i*] ≡ \$*d*[*i*]⁀{*1*} — Post-fix "yr" stands for "year".
definition *v-pres* :: *real* ⇒ *real* (\$*v*[-] [*0*] *200*)
where \$*v*[*i*] ≡ *1* / (*1*+*i*) — present value factor
definition *ann* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real* (\$*a*[-]⁀{-}′-- [*0*,*0*,*101*] *200*)
where \$*a*[*i*]⁀{*m*}-*n* ≡ ∑*k*<*n*∗*m*. \$*v*[*i*].⁀((*k*+*1*::*nat*)/*m*) / *m*
— present value of an immediate annuity

abbreviation *ann-yr* :: *real* ⇒ *nat* ⇒ *real* ($*a*[-]′-- [*0*,*101*] *200*)
  where $*a*[*i*]-*n* ≡ $*a*[*i*]^{*1*}-*n*
definition *acc* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real* ($*s*[-]^{-}′-- [*0*,*0*,*101*] *200*)
  where $*s*[*i*]^{*m*}-*n* ≡ ∑ *k*<*n*∗*m*. (*1*+*i*).^((*k*::*nat*)/*m*) / *m*
    — future value of an immediate annuity
    — The name "acc" stands for "accumulation".
abbreviation *acc-yr* :: *real* ⇒ *nat* ⇒ *real* ($*s*[-]′-- [*0*] *200*)
  where $*s*[*i*]-*n* ≡ $*s*[*i*]^{*1*}-*n*
definition *ann-due* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real* ($*a*′′′′[-]^{-}′-- [*0*,*0*,*101*] *200*)
  where $*a*′′[*i*]^{*m*}-*n* ≡ ∑ *k*<*n*∗*m*. $*v*[*i*].^((*k*::*nat*)/*m*) / *m*
    — present value of an annuity-due
abbreviation *ann-due-yr* :: *real* ⇒ *nat* ⇒ *real* ($*a*′′′′[-]′-- [*0*,*101*] *200*)
  where $*a*′′[*i*]-*n* ≡ $*a*′′[*i*]^{*1*}-*n*
definition *acc-due* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real* ($*s*′′′′[-]^{-}′-- [*0*,*0*,*101*] *200*)
  where $*s*′′[*i*]^{*m*}-*n* ≡ ∑ *k*<*n*∗*m*. (*1*+*i*).^((*k*+*1*::*nat*)/*m*) / *m*
    — future value of an annuity-due
abbreviation *acc-due-yr* :: *real* ⇒ *nat* ⇒ *real* ($*s*′′′′[-]′-- [*0*,*101*] *200*)
  where $*s*′′[*i*]-*n* ≡ $*s*′′[*i*]^{*1*}-*n*
definition *ann-cont* :: *real* ⇒ *real* ⇒ *real* ($*a*′′[-]′-- [*0*,*101*] *200*)
  where $*a*′[*i*]-*n* ≡ *integral* {*0*..*n*} (λ*t*::*real*. $*v*[*i*].^*t*)
    — present value of a continuous annuity
definition *acc-cont* :: *real* ⇒ *real* ⇒ *real* ($*s*′′[-]′-- [*0*,*101*] *200*)
  where $*s*′[*i*]-*n* ≡ *integral* {*0*..*n*} (λ*t*::*real*. (*1*+*i*).^*t*)
    — future value of a continuous annuity
definition *perp* :: *real* ⇒ *nat* ⇒ *real* ($*a*[-]^{-}′-∞ [*0*,*0*] *200*)
  where $*a*[*i*]^{*m*}-∞ ≡ *1* / $*i*[*i*]^{*m*}
    — present value of a perpetual annuity
abbreviation *perp-yr* :: *real* ⇒ *real* ($*a*[-]′-∞ [*0*] *200*)
  where $*a*[*i*]-∞ ≡ $*a*[*i*]^{*1*}-∞
definition *perp-due* :: *real* ⇒ *nat* ⇒ *real* ($*a*′′′′[-]^{-}′-∞ [*0*,*0*] *200*)
  where $*a*′′[*i*]^{*m*}-∞ ≡ *1* / $*d*[*i*]^{*m*}
    — present value of a perpetual annuity-due
abbreviation *perp-due-yr* :: *real* ⇒ *real* ($*a*′′′′[-]′-∞ [*0*] *200*)
  where $*a*′′[*i*]-∞ ≡ $*a*′′[*i*]^{*1*}-∞
definition *ann-incr* :: *nat* ⇒ *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
  ($′(*I*^{-}*a*′)[-]^{-}′-- [*0*,*0*,*0*,*101*] *200*)
  where $(*I*^{*l*}*a*)[*i*]^{*m*}-*n* ≡ ∑ *k*<*n*∗*m*. $*v*[*i*].^((*k*+*1*::*nat*)/*m*) ∗ ⌈*l*∗(*k*+*1*::*nat*)/*m*⌉ / (*l*∗*m*)
    — present value of an increasing annuity
    — This is my original definition.
    — Here, "l" represents the number of increments per unit time.
abbreviation *ann-incr-lvl* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
  ($′(*Ia*′)[-]^{-}′-- [*0*,*0*,*101*] *200*)
  where $(*Ia*)[*i*]^{*m*}-*n* ≡ $(*I*^{*1*}*a*)[*i*]^{*m*}-*n*
    — The post-fix "lvl" stands for "level".
abbreviation *ann-incr-yr* :: *real* ⇒ *nat* ⇒ *real* ($′(*Ia*′)[-]′-- [*0*,*101*] *200*)
  where $(*Ia*)[*i*]-*n* ≡ $(*Ia*)[*i*]^{*1*}-*n*
definition *acc-incr* :: *nat* ⇒ *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
  ($′(*I*^{-}*s*′)[-]^{-}′-- [*0*,*0*,*0*,*101*] *200*)

where $(I^{l}s)[i]^{m}-n \equiv \sum k<n*m.\ (1+i).^{}(n-(k+1::nat)/m) * \lceil l*(k+1::nat)/m \rceil / (l*m)$

— future value of an increasing annuity

abbreviation *acc-incr-lvl* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
($'(Is')[-]^{-}'-- [0,0,101] 200)
where $(Is)[i]^{m}-n \equiv $(I^{1}s)[i]^{m}-n$
abbreviation *acc-incr-yr* :: *real* ⇒ *nat* ⇒ *real* ($'(Is')[-]'-- [0,101] 200)
where $(Is)[i]-n \equiv $(Is)[i]^{1}-n$
definition *ann-due-incr* :: *nat* ⇒ *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
($'(I^{-}a'''')[-]^{-}'-- [0,0,0,101] 200)
where $(I^{l}a'')[i]^{m}-n \equiv \sum k<n*m.\ $v[i].^{}((k::nat)/m) * \lceil l*(k+1::nat)/m \rceil / (l*m)$
abbreviation *ann-due-incr-lvl* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
($'(Ia'''')[-]^{-}'-- [0,0,101] 200)
where $(Ia'')[i]^{m}-n \equiv $(I^{1}a'')[i]^{m}-n$
abbreviation *ann-due-incr-yr* :: *real* ⇒ *nat* ⇒ *real* ($'(Ia'''')[-]'-- [0,101] 200)
where $(Ia'')[i]-n \equiv $(Ia'')[i]^{1}-n$
definition *acc-due-incr* :: *nat* ⇒ *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
($'(I^{-}s'''')[-]^{-}'-- [0,0,0,101] 200)
where $(I^{l}s'')[i]^{m}-n \equiv \sum k<n*m.\ (1+i).^{}(n-(k::nat)/m) * \lceil l*(k+1::nat)/m \rceil / (l*m)$
abbreviation *acc-due-incr-lvl* :: *real* ⇒ *nat* ⇒ *nat* ⇒ *real*
($'(Is'''')[-]^{-}'-- [0,0,101] 200)
where $(Is'')[i]^{m}-n \equiv $(I^{1}s'')[i]^{m}-n$
abbreviation *acc-due-incr-yr* :: *real* ⇒ *nat* ⇒ *real* ($'(Is'''')[-]'-- [0,101] 200)
where $(Is'')[i]-n \equiv $(Is'')[i]^{1}-n$
definition *perp-incr* :: *nat* ⇒ *real* ⇒ *nat* ⇒ *real* ($'(I^{-}a')[-]^{-}'-∞ [0,0,0] 200)
where $(I^{l}a)[i]^{m}-∞ \equiv lim\ (λn.\ $(I^{l}a)[i]^{m}-n)$
abbreviation *perp-incr-lvl* :: *real* ⇒ *nat* ⇒ *real* ($'(Ia')[-]^{-}'-∞ [0,0] 200)
where $(Ia)[i]^{m}-∞ \equiv $(I^{1}a)[i]^{m}-∞$
abbreviation *perp-incr-yr* :: *real* ⇒ *real* ($'(Ia')[-]'-∞ [0] 200)
where $(Ia)[i]-∞ \equiv $(Ia)[i]^{1}-∞$
definition *perp-due-incr* :: *nat* ⇒ *real* ⇒ *nat* ⇒ *real* ($'(I^{-}a'''')[-]^{-}'-∞ [0,0,0] 200)
where $(I^{l}a'')[i]^{m}-∞ \equiv lim\ (λn.\ $(I^{l}a'')[i]^{m}-n)$
abbreviation *perp-due-incr-lvl* :: *real* ⇒ *nat* ⇒ *real* ($'(Ia'''')[-]^{-}'-∞ [0,0] 200)
where $(Ia'')[i]^{m}-∞ \equiv $(I^{1}a'')[i]^{m}-∞$
abbreviation *perp-due-incr-yr* :: *real* ⇒ *real* ($'(Ia'''')[-]'-∞ [0] 200)
where $(Ia'')[i]-∞ \equiv $(Ia'')[i]^{1}-∞$

3 Theory of Interest

locale *interest* =
fixes *i* :: *real* — i stands for an interest rate.
assumes *v-futr-pos*: *1 + i > 0* — Assume that the future value is positive.

context *interest*

begin

abbreviation *i-nom′* :: *nat* ⇒ *real* (\$*i*^{-} [*0*] *200*)
  where \$*i*^{*m*} ≡ \$*i*[*i*]^{*m*}
abbreviation *i-force′* :: *real* (\$*δ*)
  where \$*δ* ≡ \$*δ*[*i*]
abbreviation *d-nom′* :: *nat* ⇒ *real* (\$*d*^{-} [*0*] *200*)
  where \$*d*^{*m*} ≡ \$*d*[*i*]^{*m*}
abbreviation *d-nom-yr′* :: *real* (\$*d*)
  where \$*d* ≡ \$*d*[*i*]
abbreviation *v-pres′* :: *real* (\$*v*)
  where \$*v* ≡ \$*v*[*i*]
abbreviation *ann′* :: *nat* ⇒ *nat* ⇒ *real* (\$*a*^{-}′-- [*0*,*101*] *200*)
  where \$*a*^{*m*}-*n* ≡ \$*a*[*i*]^{*m*}-*n*
abbreviation *ann-yr′* :: *nat* ⇒ *real* (\$*a*′-- [*101*] *200*)
  where \$*a*-*n* ≡ \$*a*[*i*]-*n*
abbreviation *acc′* :: *nat* ⇒ *nat* ⇒ *real* (\$*s*^{-}′-- [*0*,*101*] *200*)
  where \$*s*^{*m*}-*n* ≡ \$*s*[*i*]^{*m*}-*n*
abbreviation *acc-yr′* :: *nat* ⇒ *real* (\$*s*′-- [*101*] *200*)
  where \$*s*-*n* ≡ \$*s*[*i*]-*n*
abbreviation *ann-due′* :: *nat* ⇒ *nat* ⇒ *real* (\$*a*′′′′^{-}′-- [*0*,*101*] *200*)
  where \$*a*′′^{*m*}-*n* ≡ \$*a*′′[*i*]^{*m*}-*n*
abbreviation *ann-due-yr′* :: *nat* ⇒ *real* (\$*a*′′′′′-- [*101*] *200*)
  where \$*a*′′-*n* ≡ \$*a*′′[*i*]-*n*
abbreviation *acc-due′* :: *nat* ⇒ *nat* ⇒ *real* (\$*s*′′′′^{-}′-- [*0*,*101*] *200*)
  where \$*s*′′^{*m*}-*n* ≡ \$*s*′′[*i*]^{*m*}-*n*
abbreviation *acc-due-yr′* :: *nat* ⇒ *real* (\$*s*′′′′′-- [*101*] *200*)
  where \$*s*′′-*n* ≡ \$*s*′′[*i*]-*n*
abbreviation *ann-cont′* :: *real* ⇒ *real* (\$*a*′′′-- [*101*] *200*)
  where \$*a*′-*n* ≡ \$*a*′[*i*]-*n*
abbreviation *acc-cont′* :: *real* ⇒ *real* (\$*s*′′′-- [*101*] *200*)
  where \$*s*′-*n* ≡ \$*s*′[*i*]-*n*
abbreviation *perp′* :: *nat* ⇒ *real* (\$*a*^{-}′-∞ [*0*] *200*)
  where \$*a*^{*m*}-∞ ≡ \$*a*[*i*]^{*m*}-∞
abbreviation *perp-yr′* :: *real* (\$*a*′-∞)
  where \$*a*-∞ ≡ \$*a*[*i*]-∞
abbreviation *perp-due′* :: *nat* ⇒ *real* (\$*a*′′′′^{-}′-∞ [*0*] *200*)
  where \$*a*′′^{*m*}-∞ ≡ \$*a*′′[*i*]^{*m*}-∞
abbreviation *perp-due-yr′* :: *real* (\$*a*′′′′′-∞)
  where \$*a*′′-∞ ≡ \$*a*′′[*i*]-∞
abbreviation *ann-incr′* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *real* (\$′(*I*^{-}*a*′)^{-}′-- [*0*,*0*,*101*] *200*)
  where \$(*I*^{*l*}*a*)^{*m*}-*n* ≡ \$(*I*^{*l*}*a*)[*i*]^{*m*}-*n*
abbreviation *ann-incr-lvl′* :: *nat* ⇒ *nat* ⇒ *real* (\$′(*Ia*′)^{-}′-- [*0*,*101*] *200*)
  where \$(*Ia*)^{*m*}-*n* ≡ \$(*Ia*)[*i*]^{*m*}-*n*
abbreviation *ann-incr-yr′* :: *nat* ⇒ *real* (\$′(*Ia*′)′-- [*101*] *200*)
  where \$(*Ia*)-*n* ≡ \$(*Ia*)[*i*]-*n*
abbreviation *acc-incr′* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *real* (\$′(*I*^{-}*s*′)^{-}′-- [*0*,*0*,*101*] *200*)

where $(I^{l}s)^{m}-n ≡ $(I^{l}s)[i]^{m}-n
abbreviation *acc-incr-lvl′* :: *nat* ⇒ *nat* ⇒ *real* ($′(*Is′*)^{-}′-- [*0*,*101*] *200*)
where $(*Is*)^{m}-n ≡ $(*Is*)[i]^{m}-n
abbreviation *acc-incr-yr′* :: *nat* ⇒ *real* ($′(*Is′*)′-- [*101*] *200*)
where $(*Is*)-n ≡ $(*Is*)[i]-n
abbreviation *ann-due-incr′* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *real* ($′(*I*^{-}*a′′′′′*)^{-}′-- [*0*,*0*,*101*] *200*)
where $(*I*^{*l*}*a′′*)^{*m*}-n ≡ $(*I*^{*l*}*a′′*)[i]^{*m*}-n
abbreviation *ann-due-incr-lvl′* :: *nat* ⇒ *nat* ⇒ *real* ($′(*Ia′′′′′*)^{-}′-- [*0*,*101*] *200*)
where $(*Ia′′*)^{*m*}-n ≡ $(*Ia′′*)[i]^{*m*}-n
abbreviation *ann-due-incr-yr′* :: *nat* ⇒ *real* ($′(*Ia′′′′′*)′-- [*101*] *200*)
where $(*Ia′′*)-n ≡ $(*Ia′′*)[i]-n
abbreviation *acc-due-incr′* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *real* ($′(*I*^{-}*s′′′′′*)^{-}′-- [*0*,*0*,*101*] *200*)
where $(*I*^{*l*}*s′′*)^{*m*}-n ≡ $(*I*^{*l*}*s′′*)[i]^{*m*}-n
abbreviation *acc-due-incr-lvl′* :: *nat* ⇒ *nat* ⇒ *real* ($′(*Is′′′′′*)^{-}′-- [*0*,*101*] *200*)
where $(*Is′′*)^{*m*}-n ≡ $(*Is′′*)[i]^{*m*}-n
abbreviation *acc-due-incr-yr′* :: *nat* ⇒ *real* ($′(*Is′′′′′*)′-- [*101*] *200*)
where $(*Is′′*)-n ≡ $(*Is′′*)[i]-n
abbreviation *perp-incr′* :: *nat* ⇒ *nat* ⇒ *real* ($′(*I*^{-}*a′*)^{-}′-∞ [*0*,*0*] *200*)
where $(*I*^{*l*}*a*)^{*m*}-∞ ≡ $(*I*^{*l*}*a*)[i]^{*m*}-∞
abbreviation *perp-incr-lvl′* :: *nat* ⇒ *real* ($′(*Ia′*)^{-}′-∞ [*0*] *200*)
where $(*Ia*)^{*m*}-∞ ≡ $(*Ia*)[i]^{*m*}-∞
abbreviation *perp-incr-yr′* :: *real* ($′(*Ia′*)′-∞)
where $(*Ia*)-∞ ≡ $(*Ia*)[i]-∞
abbreviation *perp-due-incr′* :: *nat* ⇒ *nat* ⇒ *real* ($′(*I*^{-}*a′′′′′*)^{-}′-∞ [*0*,*0*] *200*)
where $(*I*^{*l*}*a′′*)^{*m*}-∞ ≡ $(*I*^{*l*}*a′′*)[i]^{*m*}-∞
abbreviation *perp-due-incr-lvl′* :: *nat* ⇒ *real* ($′(*Ia′′′′′*)^{-}′-∞ [*0*] *200*)
where $(*Ia′′*)^{*m*}-∞ ≡ $(*Ia′′*)[i]^{*m*}-∞
abbreviation *perp-due-incr-yr′* :: *real* ($′(*Ia′′′′′*)′-∞)
where $(*Ia′′*)-∞ ≡ $(*Ia′′*)[i]-∞

lemma *v-futr-m-pos*: *1* + $*i*^{*m*}/*m* > *0* **if** *m* ≠ *0* **for** *m*::*nat*
using *v-futr-pos i-nom-def* **by** *force*

lemma *i-nom-1*[*simp*]: $*i*^{*1*} = *i*
using *v-futr-pos i-nom-def* **by** *force*

lemma *i-nom-eff*: (*1* + $*i*^{*m*}/*m*)^*m* = *1* + *i* **if** *m* ≠ *0* **for** *m*::*nat*
unfolding *i-nom-def* **using** *less-imp-neq v-futr-pos that*
apply (*simp*, *subst powr-realpow*[*THEN sym*], *simp*)
by (*subst powr-powr*, *simp*)

lemma *i-nom-i*: *1* + $*i*^{*m*}/*m* = (*1*+*i*).^(*1*/*m*) **if** *m* ≠ *0* **for** *m*::*nat*
unfolding *i-nom-def* **by** (*simp add*: *that*)

lemma *i-nom-0-iff-i-0*: $*i*^{*m*} = *0* ⟷ *i* = *0* **if** *m* ≠ *0* **for** *m*::*nat*
proof

```
  assume $i^{m} = 0
  hence ⋆: (1+i).^(1/m) = (1+i).^0
    unfolding i-nom-def using v-futr-pos that by simp
  show i = 0
  proof (rule ccontr)
    assume i ≠ 0
    hence 1/m = 0 using powr-inj ⋆ v-futr-pos by smt
    thus False using that by simp
  qed
next
  assume i = 0
  thus $i^{m} = 0
    unfolding i-nom-def by simp
qed

lemma i-nom-pos-iff-i-pos: $i^{m} > 0 ⟷ i > 0 if m ≠ 0 for m::nat
proof
  assume $i^{m} > 0
  hence ⋆: (1+i).^(1/m) > 1.^(1/m)
    unfolding i-nom-def using v-futr-pos that by (simp add: zero-less-mult-iff)
  thus i > 0
    using powr-less-cancel2[of 1/m 1 1+i] v-futr-pos that by simp
next
  assume i > 0
  hence (1+i).^(1/m) > 1.^(1/m)
    using powr-less-mono2 v-futr-pos that by simp
  thus $i^{m} > 0
    unfolding i-nom-def using that by (simp add: zero-less-mult-iff)
qed

lemma e-delta: exp $δ = 1 + i
  unfolding i-force-def by (simp add: v-futr-pos)

lemma delta-0-iff-i-0: $δ = 0 ⟷ i = 0
proof
  assume $δ = 0
  thus i = 0
    using e-delta by auto
next
  assume i = 0
  thus $δ = 0
    unfolding i-force-def by simp
qed

lemma lim-i-nom: (λm. $i^{m}) ⟶ $δ
proof −
  let ?f = λh. ((1+i).^h − 1) / h
  have D1ipwr: DERIV (λh. (1+i).^h) 0 :> $δ
    unfolding i-force-def
```

```
  using has-real-derivative-powr2[OF v-futr-pos, where x=0] v-futr-pos by simp
 hence limf: (?f ⟶ $δ) (at 0)
  unfolding DERIV-def using v-futr-pos by auto
 hence (λm. $i^{Suc m}) ⟶ $δ
  unfolding i-nom-def using tendsto-at-iff-sequentially[of ?f $δ 0 ℝ, THEN iffD1]
  apply simp
  apply (drule-tac x=λm. 1 / Suc m in spec, simp, drule mp)
  subgoal using lim-1-over-n LIMSEQ-Suc by force
  by (simp add: o-def mult.commute)
 thus ?thesis
  by (simp add: LIMSEQ-imp-Suc)
qed

lemma d-nom-0-iff-i-0: $d^{m} = 0 ⟷ i = 0 if m ≠ 0 for m::nat
proof −
 have $d^{m} = 0 ⟷ $i^{m} = 0
  unfolding d-nom-def using v-futr-m-pos by (smt (verit) divide-eq-0-iff of-nat-0)
 thus ?thesis
  using i-nom-0-iff-i-0 that by auto
qed

lemma d-nom-pos-iff-i-pos: $d^{m} > 0 ⟷ i > 0 if m ≠ 0 for m::nat
proof −
 have $d^{m} > 0 ⟷ $i^{m} > 0
  unfolding d-nom-def using zero-less-divide-iff i-nom-pos-iff-i-pos v-futr-m-pos
that by smt
 thus ?thesis
  using i-nom-pos-iff-i-pos that by auto
qed

lemma d-nom-i-nom: 1 − $d^{m}/m = 1 / (1 + $i^{m}/m) if m ≠ 0 for m::nat
proof −
 have 1 − $d^{m}/m = 1 − ($i^{m}/m) / (1 + $i^{m}/m)
  by (simp add: d-nom-def)
 also have ... = 1 / (1 + $i^{m}/m)
  using v-futr-m-pos
  by (smt (verit, ccfv-SIG) add-divide-distrib that div-self)
 finally show ?thesis .
qed

lemma lim-d-nom: (λm. $d^{m}) ⟶ $δ
proof −
 have (λm. $i^{m}/m) ⟶ 0
  using lim-i-nom tendsto-divide-0 tendsto-of-nat by blast
 hence (λm. 1 + $i^{m}/m) ⟶ 1
  by (metis add.right-neutral tendsto-add-const-iff)
 thus ?thesis
  unfolding d-nom-def using lim-i-nom tendsto-divide div-by-1 by fastforce
```

qed

lemma *v-pos*: $\$v > 0$
unfolding *v-pres-def* **using** *v-futr-pos* **by** *auto*

lemma *v-1-iff-i-0*: $\$v = 1 \longleftrightarrow i = 0$
proof
assume $\$v = 1$
thus $i = 0$
unfolding *v-pres-def* **by** *simp*
next
assume $i = 0$
thus $\$v = 1$
unfolding *v-pres-def* **by** *simp*
qed

lemma *v-lt-1-iff-i-pos*: $\$v < 1 \longleftrightarrow i > 0$
proof
assume $\$v < 1$
thus $i > 0$
unfolding *v-pres-def* **by** (*simp add*: *v-futr-pos*)
next
assume $i > 0$
thus $\$v < 1$
unfolding *v-pres-def* **by** (*simp add*: *v-futr-pos*)
qed

lemma *v-i-nom*: $\$v = (1 + \$i\hat{}\{m\}/m).\hat{}-m$ **if** $m \neq 0$ **for** *m*::*nat*
proof −
have $\$v = (1 + i).\hat{}-1$
unfolding *v-pres-def* **using** *v-futr-pos powr-real-def that* **by** (*simp add*: *powr-neg-one*)
also have $\ldots = ((1 + \$i\hat{}\{m\}/m)\hat{}m).\hat{}-1$
using *i-nom-eff that* **by** *presburger*
also have $\ldots = (1 + \$i\hat{}\{m\}/m).\hat{}-m$
using *powr-powr powr-realpow*[*THEN sym*] *v-futr-m-pos that* **by** *simp*
finally show *?thesis* **.**
qed

lemma *i-v*: $1 + i = \$v.\hat{}-1$
unfolding *v-pres-def powr-real-def* **using** *v-futr-pos powr-neg-one* **by** *simp*

lemma *i-v-powr*: $(1 + i).\hat{}a = \$v.\hat{}-a$ **for** *a*::*real*
by (*subst i-v*, *subst powr-powr*, *simp*)

lemma *v-delta*: $ln\ \$v = -\ \$\delta$
unfolding *i-force-def v-pres-def* **using** *v-futr-pos* **by** (*simp add*: *ln-div*)

lemma *is-derive-vpow*: $DERIV\ (\lambda t.\ \$v.\hat{}t)\ t :> -\ \$\delta * \$v.\hat{}t$
using *v-delta has-real-derivative-powr2 v-pos* **by** (*metis mult.commute*)

lemma *d-nom-v*: $d^{m} = m * (1 - $v.^(1/m)) **if** $m \neq 0$ **for** *m*::*nat*
proof −
  have $d^{m} = m * (1 - 1 / (1 + $i^{m}/m))
    using *d-nom-i-nom*[*THEN sym*] *that* **by** *force*
  also have ... = $m * (1 - 1 / (1 + i).^(1/m))$
    using *i-nom-i that powr-minus-divide* **by** *simp*
  also have ... = $m * (1 - $v.^(1/m))$
    using *v-pres-def v-futr-pos powr-divide* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *d-nom-i-nom-v*: $d^{m} = $i^{m} * $v.^(1/m)$ **if** $m \neq 0$ **for** *m*::*nat*
  unfolding *d-nom-def v-pres-def* **using** *i-nom-i powr-divide v-futr-pos that* **by** *auto*

lemma *a-calc*: $a^{m}-n = (1 - $v^n) / $i^{m} **if** $m \neq 0$ $i \neq 0$ **for** *n m* ::*nat*
proof −
  have $\bigwedge l$::*nat*. $l/m = (1/m) * l$ **by** *simp*
  hence $\star$: $\bigwedge l$::*nat*. $v.^(l/m) = ($v.^(1/m))^l$
    using *powr-powr powr-realpow v-pos* **by** (*metis powr-gt-zero*)
  hence $a^{m}-n = (\sum k<n*m. ($v.^(1/m))^(k+1::nat) / m)$
    unfolding *ann-def* **by** *presburger*
  also have ... = $v.^(1/m) * (\sum k<n*m. ($v.^(1/m))^k) / m$
    by (*simp*, *subst sum-divide-distrib*[*THEN sym*], *subst sum-distrib-left*[*THEN sym*], *simp*)
  also have ... = $v.^(1/m) * ((($v.^(1/m))^(n*m) - 1) / ($v.^(1/m) - 1)) / m$
    apply (*subst geometric-sum*[*of* $v.^(1/m) n*m$]; *simp?*)
    using *powr-zero-eq-one*[*of* v] *v-pos v-1-iff-i-0 powr-inj that*
    by (*smt* (*verit*, *del-insts*) *divide-eq-0-iff of-nat-eq-0-iff*)
 also have ... = (($v.^(1/m))^(n*m) - 1) / (m * ($v.^(1/m) - 1) / $v.^(1/m))$
    by (*simp add*: *field-simps*)
  also have ... = ($v^n - 1) / (m * (1 - 1 / $v.^(1/m)))$
    apply (*subst* $\star$[*of n*m::nat*, *THEN sym*], *simp only*: *of-nat-simps*)
    apply (*subst nonzero-mult-div-cancel-right*[**where** *'a=real*, *of m n*], *simp add*: *that*)
    apply (*subst powr-realpow*[*OF v-pos*])
    apply (*subst times-divide-eq-right*[*of - - $v.^(1/m)*, *THEN sym*])
    using *v-pos* **by** (*subst diff-divide-distrib*[*of - - $v.^(1/m)*], *simp*)
  also have ... = (1 - $v^n) / (m * (1 / $v.^(1/m) - 1))$
    using *minus-divide-divide* **by** (*smt mult-minus-right*)
  also have ... = (1 - $v^n) / i^{m}
    unfolding *i-nom-def v-pres-def* **using** *v-futr-pos powr-divide* **by** *auto*
  finally show *?thesis* **.**
qed

lemma *a-calc-i-0*: $a^{m}-n = n$ **if** $m \neq 0$ $i = 0$ **for** *n m* :: *nat*
  unfolding *ann-def v-pres-def* **using** *that* **by** *simp*

lemma *s-calc-i-0*: $s^{m}-n = n **if** m ≠ 0 i = 0 **for** n m :: nat
unfolding *acc-def* **using** *that* **by** *simp*

lemma *s-a*: $s^{m}-n = (1+i)^n * $a^{m}-n **if** m ≠ 0 **for** n m :: nat
proof −
have (1+i)^n * $a^{m}-n = (∑ k<n*m. (1+i)^n * ($v.^((k+1::nat)/m) / m))
unfolding *ann-def* **using** *sum-distrib-left* **by** *blast*
also have ... = (∑ k<n*m. (1+i).^((n*m − Suc k)/m) / m)
proof −
have ⋀k::nat. k < n*m ⟹ (1+i)^n * ($v.^((k+1::nat)/m) / m) = (1+i).^((n*m − Suc k)/m) / m
unfolding *v-pres-def*
apply (*subst powr-realpow*[*THEN sym*], *simp add*: *v-futr-pos*)
apply (*subst inverse-powr*, *simp add*: *v-futr-pos*)
apply (*subst times-divide-eq-right*, *subst powr-add*[*THEN sym*], *simp add*: *that*)
by (*subst of-nat-diff*, *simp add*: *Suc-le-eq*, *simp add*: *diff-divide-distrib that*)
thus *?thesis* **by** (*meson lessThan-iff sum.cong*)
qed
also have ... = (∑ k<n*m. (1+i).^(k/m) / m)
apply (*subst atLeast0LessThan*[*THEN sym*])+
by (*subst sum.atLeastLessThan-rev*[*THEN sym*, *of* - n*m 0, *simplified add-0-right*], *simp*)
also have ... = $s^{m}-n
unfolding *acc-def* **by** *simp*
finally show *?thesis* **..**
qed

lemma *s-calc*: $s^{m}-n = ((1+i)^n − 1) / $i^{m} **if** m ≠ 0 i ≠ 0 **for** n m :: nat
using *that v-futr-pos*
apply (*subst s-a*, *simp*, *subst a-calc*; *simp?*)
apply (*rule disjI2*)
apply (*subst right-diff-distrib*, *simp*)
apply (*rule left-right-inverse-power*)
unfolding *v-pres-def* **by** *auto*

lemma *a''-a*: $a''^{m}-n = (1+i).^(1/m) * $a^{m}-n **if** m ≠ 0 **for** m::nat
unfolding *ann-def ann-due-def*
apply (*subst sum-distrib-left*, *subst times-divide-eq-right*, *simp*)
by (*subst i-v*, *subst powr-powr*, *subst powr-add*[*THEN sym*], *simp*, *subst add-divide-distrib*, *simp*)

lemma *a-a''*: $a^{m}-n = $v.^(1/m) * $a''^{m}-n **if** m ≠ 0 **for** m::nat
unfolding *ann-def ann-due-def*
apply (*subst sum-distrib-left*, *subst times-divide-eq-right*, *simp*)
by (*subst powr-add*[*THEN sym*], *subst add-divide-distrib*, *simp*)

lemma *a''-calc-i-0*: $a''^{m}-n = n **if** $m \neq 0$ $i = 0$ **for** n m :: *nat*
unfolding *ann-due-def v-pres-def* **using** *that* **by** *simp*

lemma *s''-calc-i-0*: $s''^{m}-n = n **if** $m \neq 0$ $i = 0$ **for** n m :: *nat*
unfolding *acc-due-def* **using** *that* **by** *simp*

lemma *a''-calc*: $a''^{m}-n = (1 - $v^n) / $d^{m} **if** $m \neq 0$ $i \neq 0$ **for** n m :: *nat*
proof −
have $a''^{m}-n = (1+i).^(1/m) * ((1 - $v^n) / $i^{m})
using *a''-a a-calc times-divide-eq-right that* **by** *simp*
also have ... = (1 - $v^n) / ($v.^(1/m) * $i^{m})
by (*subst i-v*, *subst powr-powr*, *simp*, *subst powr-minus-divide*, *simp*)
also have ... = (1 - $v^n) / $d^{m}
using *d-nom-i-nom-v that* **by** *simp*
finally show *?thesis* **.**
qed

lemma *s''-s*: $s''^{m}-n = (1+i).^(1/m) * $s^{m}-n **if** $m \neq 0$ **for** *m*::*nat*
unfolding *acc-def acc-due-def*
by (*simp add*: *sum-distrib-left add-divide-distrib powr-add*)

lemma *s-s''*: $s^{m}-n = $v.^(1/m) * $s''^{m}-n **if** $m \neq 0$ **for** *m*::*nat*
unfolding *acc-def acc-due-def v-pres-def* **using** *v-futr-pos*
apply (*simp add*: *sum-distrib-left inverse-powr add-divide-distrib*)
by (*metis* (*no-types*) *add-diff-cancel-left' powr-add uminus-add-conv-diff*)

lemma *s''-calc*: $s''^{m}-n = ((1+i)^n - 1) / $d^{m} **if** $m \neq 0$ $i \neq 0$ **for** n m :: *nat*
proof −
have $s''^{m}-n = (1+i).^(1/m) * ((1+i)^n - 1) / $i^{m}
using *s''-s s-calc times-divide-eq-right that* **by** *simp*
also have ... = ((1+i)^n - 1) / ($v.^(1/m) * $i^{m})
by (*subst i-v*, *subst powr-powr*, *simp*, *subst powr-minus-divide*, *simp*)
also have ... = ((1+i)^n - 1) / $d^{m}
using *d-nom-i-nom-v that* **by** *simp*
finally show *?thesis* **.**
qed

lemma *s''-a''*: $s''^{m}-n = (1+i)^n * $a''^{m}-n **if** $m \neq 0$ **for** *m*::*nat*
using *that s''-s a''-a s-a* **by** *simp*

lemma *a'-calc*: $a'-n = (1 - $v.^n) / δ **if** $i \neq 0$ $n \geq 0$ **for** *n*::*real*
unfolding *ann-cont-def*
apply (*rule integral-unique*)
using *has-integral-powr2-from-0*[*OF v-pos - that*(*2*)] *v-delta v-1-iff-i-0 that*
by (*smt minus-divide-divide*)

lemma *a'-calc-i-0*: $a'-n = n **if** $i = 0$ $n \geq 0$ **for** *n*::*real*

```
  unfolding ann-cont-def
  apply (subst iffD2[OF v-1-iff-i-0], simp add: that)
  by (simp add: integral-cong that)

lemma s'-calc: $s'-n = ((1+i).^n - 1) / $δ if i ≠ 0 n ≥ 0 for n::real
  unfolding acc-cont-def
  apply (rule integral-unique)
  using has-integral-powr2-from-0[OF v-futr-pos - that(2)] i-force-def that
  by simp

lemma s'-calc-i-0: $s'-n = n if i = 0 n ≥ 0 for n::real
  unfolding acc-cont-def
  apply (subst ‹i = 0›, simp)
  by (simp add: integral-cong that)

lemma s'-a': $s'-n = (1+i).^n * $a'-n if n ≥ 0 for n::real
proof −
  have (1+i).^n * $a'-n = integral {0..n} (λt. (1+i).^(n−t))
    unfolding ann-cont-def
    using integrable-on-powr2-from-0-general[of $v n] v-pos v-futr-pos that
    apply (subst integral-mult, simp)
    apply (rule integral-cong)
    unfolding v-pres-def using inverse-powr powr-add[THEN sym] by smt
  also have ... = $s'-n
    unfolding acc-cont-def using v-futr-pos that
   apply (subst has-integral-interval-reverse[of 0 n, simplified, THEN integral-unique];
simp?)
    by (rule continuous-on-powr; auto)
  finally show ?thesis ..
qed

lemma lim-m-a: (λm. $a^{m}-n) ⟶ $a'-n for n::nat
proof (rule LIMSEQ-imp-Suc)
  show (λm. $a^{Suc m}-n) ⟶ $a'-n
  proof (cases i = 0)
    case True
    show ?thesis
      using a-calc-i-0 a'-calc-i-0 True by simp
  next
    case False
    show ?thesis
      using False v-pos delta-0-iff-i-0
      apply (subst a-calc; simp?)
      apply (subst a'-calc; simp?)
      apply (subst powr-realpow, simp)
      apply (rule tendsto-divide; simp?)
      by (rule LIMSEQ-Suc[OF lim-i-nom])
  qed
qed
```

lemma *lim-m-a''*: $(\lambda m.\ \$a''\hat{}\{m\}\text{-}n) \longrightarrow \$a'\text{-}n$ **for** *n*::*nat*
proof (*rule LIMSEQ-imp-Suc*)
  show $(\lambda m.\ \$a''\hat{}\{Suc\ m\}\text{-}n) \longrightarrow \$a'\text{-}n$
  proof (*cases i = 0*)
    case *True*
    show *?thesis*
      using *a''-calc-i-0 a'-calc-i-0 True* **by** *simp*
  next
    case *False*
    show *?thesis*
      using *False v-pos delta-0-iff-i-0*
      apply (*subst a''-calc*; *simp?*)
      apply (*subst a'-calc*; *simp?*)
      apply (*subst powr-realpow*, *simp*)
      apply (*rule tendsto-divide*; *simp?*)
      by (*rule LIMSEQ-Suc*[*OF lim-d-nom*])
  qed
qed

lemma *lim-m-s*: $(\lambda m.\ \$s\hat{}\{m\}\text{-}n) \longrightarrow \$s'\text{-}n$ **for** *n*::*nat*
proof (*rule LIMSEQ-imp-Suc*)
  show $(\lambda m.\ \$s\hat{}\{Suc\ m\}\text{-}n) \longrightarrow \$s'\text{-}n$
  proof (*cases i = 0*)
    case *True*
    show *?thesis*
      using *s-calc-i-0 s'-calc-i-0 True* **by** *simp*
  next
    case *False*
    show *?thesis*
      using *False v-futr-pos delta-0-iff-i-0*
      apply (*subst s-calc*; *simp?*)
      apply (*subst s'-calc*; *simp?*)
      apply (*subst powr-realpow*, *simp*)
      apply (*rule tendsto-divide*; *simp?*)
      by (*rule LIMSEQ-Suc*[*OF lim-i-nom*])
  qed
qed

lemma *lim-m-s''*: $(\lambda m.\ \$s''\hat{}\{m\}\text{-}n) \longrightarrow \$s'\text{-}n$ **for** *n*::*nat*
proof (*rule LIMSEQ-imp-Suc*)
  show $(\lambda m.\ \$s''\hat{}\{Suc\ m\}\text{-}n) \longrightarrow \$s'\text{-}n$
  proof (*cases i = 0*)
    case *True*
    show *?thesis*
      using *s''-calc-i-0 s'-calc-i-0 True* **by** *simp*
  next
    case *False*
    show *?thesis*

```
    using False v-futr-pos delta-0-iff-i-0
    apply (subst s''-calc; simp?)
    apply (subst s'-calc; simp?)
    apply (subst powr-realpow, simp)
    apply (rule tendsto-divide; simp?)
    by (rule LIMSEQ-Suc[OF lim-d-nom])
  qed
qed

lemma lim-n-a: (λn. $a^{m}-n) ⟶ $a^{m}-∞ if m ≠ 0 i > 0 for m::nat
proof −
  have $i^{m} ≠ 0 using i-nom-pos-iff-i-pos that by smt
  moreover have (λn. $v^n) ⟶ 0
    using LIMSEQ-realpow-zero[of $v] v-pos v-lt-1-iff-i-pos that by simp
  ultimately show ?thesis
    using that apply (subst a-calc; simp?)
    unfolding perp-def apply (rule tendsto-divide; simp?)
    using tendsto-diff[where a=1 and b=0] by auto
qed

lemma lim-n-a'': (λn. $a''^{m}-n) ⟶ $a''^{m}-∞ if m ≠ 0 i > 0 for m::nat
proof −
  have $d^{m} ≠ 0 using d-nom-pos-iff-i-pos that by smt
  moreover have (λn. $v^n) ⟶ 0
    using LIMSEQ-realpow-zero[of $v] v-pos v-lt-1-iff-i-pos that by simp
  ultimately show ?thesis
    using that apply (subst a''-calc; simp?)
    unfolding perp-due-def apply (rule tendsto-divide; simp?)
    using tendsto-diff[where a=1 and b=0] by auto
qed

lemma Ilsm-Ilam: $(I^{l}s)^{m}-n = (1+i)^n * $(I^{l}a)^{m}-n
  if l ≠ 0 m ≠ 0 for l n m :: nat
  unfolding acc-incr-def ann-incr-def v-pres-def using v-futr-pos powr-realpow
  apply (subst inverse-powr, simp)
  apply (subst sum-distrib-left)
 by (subst minus-real-def, subst powr-add, subst times-divide-eq-right, subst mult.assoc,
simp)

lemma Iam-calc: $(Ia)^{m}-n = (∑ j<n. (j+1)/m * (∑ k=j*m..<(j+1)*m. $v.^((k+1)/m)))
  if m ≠ 0 for n m :: nat
proof −
  let ?I = {..<n}
  let ?A = λj. {j*m..<(j+1)*m}
  let ?g = λk. $v.^((k+1::nat)/m) * ⌈(k+1::nat)/m⌉ / m
 have $(Ia)^{m}-n = (∑ j<n. ∑ k=j*m..<(j+1)*m. $v.^((k+1)/m) * ⌈(k+1)/m⌉
/ m)
    unfolding ann-incr-def using seq-part-multiple that
    apply (simp only: mult-1)
```

```
    by (subst sum.UNION-disjoint[of ?I ?A ?g, THEN sym]; simp)
  also have ... = (∑j<n. (j+1)/m * (∑ k=j*m..<(j+1)*m. $v.^((k+1)/m)))
  proof −
    { fix j k
      assume j*m ≤ k ∧ k < (j+1)*m
      hence j*m < k+1 ∧ k+1 ≤ (j+1)*m by force
      hence j < (k+1)/m ∧ (k+1)/m ≤ j+1
        using pos-less-divide-eq pos-divide-le-eq of-nat-less-iff of-nat-le-iff that
        by (smt (verit) of-nat-le-0-iff of-nat-mult)
      hence ⌈(k+1)/m⌉ = j+1
        by (simp add: ceiling-unique) }
    hence ⋀j k. j*m ≤ k ∧ k < (j+1)*m ⟹ ⌈(k+1)/m⌉ = j+1
      by (metis (no-types) of-nat-1 of-nat-add)
    with v-pos show ?thesis
      apply (intro sum.cong, simp)
      apply (subst sum-distrib-left, rule sum.cong; simp)
      by (smt (verit, ccfv-SIG) of-int-1 of-int-diff of-int-of-nat-eq)
  qed
  finally show ?thesis .
qed

lemma Ism-calc: $(Is)^{m}-n = (∑j<n. (j+1)/m * (∑ k=j*m..<(j+1)*m. (1+i).^(n−(k+1)/m)))
  if m ≠ 0 for n m :: nat
  using v-pos that
  apply (subst Ilsm-Ilam; simp)
  apply (subst Iam-calc[simplified]; simp?)
  apply ((subst sum-distrib-left, rule sum.cong; simp))+
  unfolding v-pres-def using v-futr-pos
  apply (subst inverse-powr; simp)
  apply (subst powr-realpow[THEN sym], simp)
  by (subst powr-add[THEN sym]; simp)

lemma Imam-calc-aux: $(I^{m}a)^{m}-n = (∑ k<n*m. $v.^((k+1)/m) * (k+1)
/ m^2)
  if m ≠ 0 for m::nat
  unfolding ann-incr-def power-def
  apply (rule sum.cong, simp)
  apply (subst of-nat-mult)
  using v-pos that
  apply (subst nonzero-mult-div-cancel-left, simp)
  by (subst ceiling-of-nat; simp)

lemma Imam-calc:
  $(I^{m}a)^{m}-n = ($v.^(1/m) * (1 − (n*m+1)*$v^n + n*m*$v.^(n+1/m)))
/ (m*(1−$v.^(1/m)))^2
  if i ≠ 0 m ≠ 0 for n m :: nat
proof −
  have ⋆: $v.^(1/m) > 0 using v-pos by force
  hence $(I^{m}a)^{m}-n = (∑ k<n*m. (k+1)*($v.^(1/m))^(k+1)) / m^2
```

using *that*
apply (*subst Imam-calc-aux*, *simp*)
apply (*subst sum-divide-distrib*[*THEN sym*], *simp*)
apply (*rule sum.cong*; *simp*)
using *powr-realpow*[*THEN sym*] *powr-powr* **by** (*simp add*: *add-divide-distrib powr-add*)
also have ... = $v.^(1/m) * (∑k<n*m. (k+1)*($v.^(1/m))^k) / m^2
by (*subst sum-distrib-left*, *simp add*: *that*, *rule sum.cong*; *simp*)
also have ... = $v.^(1/m) *
((1 − (n*m+1)*($v.^(1/m))^(n*m) + n*m*($v.^(1/m))^(n*m+1)) / (1 − $v.^(1/m))^2) / m^2
using *v-pos v-1-iff-i-0 that* **by** (*subst geometric-increasing-sum*; *simp?*)
also have ... = ($v.^(1/m) * (1 − (n*m+1)*$v^n + n*m*$v.^(n+1/m))) / (m*(1−$v.^(1/m)))^2
using ⋆
apply (*subst powr-realpow*[*of* $v.^(1/m), *THEN sym*], *simp*)+
apply (*subst powr-powr*)+
apply (*subst times-divide-eq-right*[*THEN sym*], *subst divide-divide-eq-left*)
apply (*subst power-mult-distrib*)
using *powr-eq-one-iff-gen v-pos v-1-iff-i-0* **apply** (*simp add*: *field-simps*)
by ((*subst powr-realpow*, *simp*)+, *simp*)
finally show *?thesis* **.**
qed

lemma *Imam-calc-i-0*: $(I^{m}a)^{m}-n = (n*m+1)*n / (2*m) **if** *i = 0 m ≠ 0*
for *n m* :: *nat*
proof −
have $(I^{m}a)^{m}-n = (∑k<n*m. $v.^((k+1)/m) * (k+1) / m^2)
by (*subst Imam-calc-aux*, *simp-all add*: *that*)
also have ... = (∑k<n*m. k+1) / m^2
apply (*subst v-1-iff-i-0*[*THEN iffD2*], *simp-all add*: *that*)
by (*subst sum-divide-distrib*[*THEN sym*], *simp*)
also have ... = (n*m*(n*m+1) *div* 2) / m^2
apply (*subst Suc-eq-plus1*[*THEN sym*], *subst sum-bounds-lt-plus1*[*of id*, *simplified*])
by (*subst Sum-Icc-nat*, *simp*)
also have ... = (n*m+1)*n / (2*m)
apply (*subst real-of-nat-div*, *simp*)
using *that* **by** (*subst power2-eq-square*, *simp add*: *field-simps*)
finally show *?thesis* **.**
qed

lemma *Imsm-calc*:
$(I^{m}s)^{m}-n = ((1+i).^(n+1/m) − (n*m+1)*(1+i).^(1/m) + n*m) / (m*((1+i).^(1/m)−1))^2
if *i ≠ 0 m ≠ 0* **for** *n m* :: *nat*
proof −
have $(I^{m}a)^{m}-n =
($v^n * ((1+i).^(n+1/m) − (n*m+1)*(1+i).^(1/m) + n*m)) / (m*((1+i).^(1/m)−1))^2

```
  proof −
    have $(I^{m}a)^{m}-n =
    ($v.^(1/m) * (1 − (n*m+1)*$v^n + n*m*$v.^(n+1/m))) / (m*(1−$v.^(1/m)))^2
      using that by (subst Imam-calc; simp)
   also have ... = (1 − (n*m+1)*$v^n + n*m*$v.^(n+1/m)) / ($v.^(1/m)*(m*($v.^(−1/m)−1))^2)
      apply (subgoal-tac $v.^(−1/m) = 1 / $v.^(1/m), erule ssubst)
       apply ((subst power2-eq-square)+, simp add: field-simps that)
      by (simp add: powr-minus-divide)
    also have ... =
          ($v.^(n+1/m) * ($v.^(−n−1/m) − (n*m+1)*$v.^(−1/m) + n*m)) /
($v.^(1/m)*(m*($v.^(−1/m)−1))^2)
      apply (subgoal-tac $v.^(−n−1/m) = 1 / $v.^(n+1/m) $v.^(−1/m) = $v^n
/ $v.^(n+1/m))
        apply ((erule ssubst)+, simp-all add: field-simps)
      using v-pos
       apply (simp add: powr-diff[THEN sym] powr-realpow[THEN sym])
      by (smt powr-minus-divide)
    also have ... =
    ($v^n * ($v.^(−n−1/m) − (n*m+1)*$v.^(−1/m) + n*m)) / ((m*($v.^(−1/m)−1))^2)
      apply (subst powr-add[of - n 1/m])
      using v-pos powr-realpow by simp
    also have ... =
    ($v^n * ((1+i).^(n+1/m) − (n*m+1)*(1+i).^(1/m) + n*m)) / ((m*((1+i).^(1/m)−1))^2)
     apply (subgoal-tac −n−1/m = −(n+1/m) −1/m = −(1/m), (erule ssubst)+)
        apply (subst i-v-powr[THEN sym])+
      by simp-all
    finally show ?thesis .
  qed
  thus ?thesis
    apply −
    using that v-futr-pos
    apply (subst Ilsm-Ilam, simp)
    apply (erule ssubst, simp)
    apply (rule disjI2)
    by (subst power-mult-distrib[THEN sym], simp add: v-pres-def)
qed

lemma Imsm-calc-i-0: $(I^{m}s)^{m}-n = (n*m+1)*n / (2*m) if i = 0 m ≠ 0
for n m :: nat
  using that
  apply (subst Ilsm-Ilam, simp)
  by (subst Imam-calc-i-0; simp)

lemma Ila''m-Ilam: $(I^{l}a'')^{m}-n = (1+i).^(1/m) * $(I^{l}a)^{m}-n
  if l ≠ 0 m ≠ 0 for l m n :: nat
  unfolding ann-incr-def ann-due-incr-def using that
  apply (subst i-v, subst powr-powr, simp)
  apply (subst sum-distrib-left)
  apply (rule sum.cong; simp)
```

apply (*rule disjI2*)
by (*smt* (*verit*) *add-divide-distrib powr-add*)

lemma *Ia''m-calc*: $(Ia'')^{m}-n = (\sum j<n.\ (j+1)/m * (\sum k=j*m..<(j+1)*m.\ \$v.^(k/m)))$
if $m \neq 0$ **for** *n m* :: *nat*
using *that*
apply (*subst Ila''m-Ilam*; *simp del*: *One-nat-def*)
apply (*subst Iam-calc*; *simp*)
apply (*subst sum-distrib-left*)
apply (*rule sum.cong*; *simp*)
apply (*subst sum-distrib-left*)+
apply (*rule sum.cong*; *simp*)
apply (*subst i-v-powr*)
using *powr-add*[*of* v, *THEN sym*] **by** (*simp add*: *field-simps*)

lemma *Ima''m-calc-aux*: $(I^{m}a'')^{m}-n = (\sum k<n*m.\ \$v.^(k/m) * (k+1) / m^2)$
if $m \neq 0$ **for** *m*::*nat*
using *that*
apply (*subst Ila''m-Ilam*, *simp*)
apply (*subst Imam-calc-aux*, *simp*)
apply (*subst sum-distrib-left*)
apply (*rule sum.cong*; *simp*)
using *powr-add*[*of* v, *THEN sym*] *i-v-powr* **by** (*simp add*: *field-simps*)

lemma *Ima''m-calc*: $(I^{m}a'')^{m}-n = (1 - (n*m+1)*\$v^n + n*m*\$v.^(n+1/m)) / (m*(1-\$v.^(1/m)))^2$
if $i \neq 0$ $m \neq 0$ **for** *n m* :: *nat*
using *that v-pos*
apply (*subst Ila''m-Ilam*, *simp*)
apply (*subst Imam-calc*; *simp*)
by (*smt* (*verit*, *del-insts*) *i-v-powr powr-add powr-zero-eq-one*)

lemma *Ils''m-Ilsm*: $(I^{l}s'')^{m}-n = (1+i).^(1/m) * (I^{l}s)^{m}-n$
if $l \neq 0$ $m \neq 0$ **for** *l m n* :: *nat*
unfolding *acc-incr-def acc-due-incr-def sum-distrib-left* **using** *that*
apply (*intro sum.cong*; *simp*)
by (*smt* (*verit*, *ccfv-SIG*) *add-divide-distrib powr-add*)

lemma *Ims''m-calc*:
$(I^{m}s'')^{m}-n =$
$(1+i).^(1/m) * ((1+i).^(n+1/m) - (n*m+1)*(1+i).^(1/m) + n*m) / (m*((1+i).^(1/m)-1))^2$
if $i \neq 0$ $m \neq 0$ **for** *n m* :: *nat*
using *that* **by** (*simp add*: *Ils''m-Ilsm Imsm-calc*)

lemma *lim-Imam*: $(\lambda n.\ \$(I^{m}a)^{m}-n) \longrightarrow 1 / (\$i^{m}*\$d^{m})$ **if** $m \neq 0$ $i > 0$ **for** *m*::*nat*

proof −
have $(\lambda n.\ \$(I\hat{}\{m\}a)\hat{}\{m\}\text{-}n) =$
$(\lambda n.\ \$v.\hat{}(1/m) * (1 - (n*m+1)*\$v\hat{}n + n*m*\$v.\hat{}(n+1/m)) / (m*(1-\$v.\hat{}(1/m)))\hat{}2)$
using *that* **by** (*subst Imam-calc*; *simp*)
moreover have $(\lambda n.\ \$v.\hat{}(1/m) * (1 - (n*m+1)*\$v\hat{}n + n*m*\$v.\hat{}(n+1/m))$
$/ (m*(1-\$v.\hat{}(1/m)))\hat{}2)$
$\longrightarrow 1 / (\$i\hat{}\{m\}*\$d\hat{}\{m\})$
proof −
have $\star$: $|\$v| < 1$
using *v-lt-1-iff-i-pos v-pos that* **by** *force*
hence $(\lambda n.\ (n*m+1)*\$v\hat{}n) \longrightarrow 0$
apply (*subst tendsto-cong*[*of* - $(\lambda n.\ n*m*\$v\hat{}n + \$v\hat{}n)$])
apply (*rule always-eventually*, *rule allI*)
apply (*simp add*: *distrib-right*)
apply (*subgoal-tac 0 = 0 + 0*, *erule ssubst*, *intro tendsto-intros*; *simp*)
apply (*subst mult.commute*, *subst mult.assoc*)
apply (*subgoal-tac 0 = real m * 0*, *erule ssubst*, *intro tendsto-intros*; *simp?*)
by (*rule powser-times-n-limit-0*; *simp*)
moreover have $(\lambda n.\ n*m*\$v.\hat{}(n+1/m)) \longrightarrow 0$
apply (*subst tendsto-cong*[*of* - $(\lambda n.\ (m*\$v.\hat{}(1/m))*(n*\$v\hat{}n))$])
apply (*rule always-eventually*, *rule allI*)
apply (*simp add*: *powr-add powr-realpow v-pos*)
apply (*subgoal-tac 0 = m*\$v.^(1/m) * 0*, *erule ssubst*, *intro tendsto-intros*;
simp?)
by (*rule powser-times-n-limit-0*, *simp add*: $\star$)
ultimately have $(\lambda n.\ \$v.\hat{}(1/m) * (1 - (n*m+1)*\$v\hat{}n + n*m*\$v.\hat{}(n+1/m))$
$/ (m*(1-\$v.\hat{}(1/m)))\hat{}2)$
$\longrightarrow \$v.\hat{}(1/m) * (1 - 0 + 0)/ (m*(1-\$v.\hat{}(1/m)))\hat{}2$
using *v-lt-1-iff-i-pos v-pos that* **by** (*intro tendsto-intros*; *simp*)
thus *?thesis*
unfolding *i-nom-def* **using** *v-pos that*
apply (*subst i-v-powr*, *subst powr-minus-divide*, *subst d-nom-v*; *simp*)
by (*subst*(*asm*)(*2*) *power2-eq-square*, *simp add*: *field-simps*)
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *perp-incr-calc*: $\$(I\hat{}\{m\}a)\hat{}\{m\}\text{-}\infty = 1 / (\$i\hat{}\{m\}*\$d\hat{}\{m\})$ **if** $m \neq 0$ $i >$
0 **for** m::*nat*
unfolding *perp-incr-def* **by** (*rule limI*, *rule lim-Imam*; *simp add*: *that*)

lemma *lim-Ima''m*: $(\lambda n.\ \$(I\hat{}\{m\}a'')\hat{}\{m\}\text{-}n) \longrightarrow 1 / (\$d\hat{}\{m\})\hat{}2$ **if** $m \neq 0$
$i > 0$ **for** m::*nat*
unfolding *perp-due-incr-def* **using** *that*
apply (*subst Ila''m-Ilam*, *simp*, *subst mult.commute*, *subst i-v-powr*, *subst powr-minus-divide*)
apply (*subgoal-tac 1/*($d\hat{}\{m\}$)$\hat{}2 = (1/(\$i\hat{}\{m\}*\$d\hat{}\{m\}))*(1/\$v.\hat{}(1/m))$, *erule*
ssubst)
apply (*intro tendsto-intros*, *simp add*: *lim-Imam*)
by (*simp add*: *d-nom-i-nom-v power2-eq-square*)

lemma *perp-due-incr-calc*: $(I^{m}a'')^{m}-\infty = 1 / (\$d^{m})^2$ **if** $m \neq 0$ $i > 0$ **for** m::*nat*
unfolding *perp-due-incr-def* **by** (*rule limI*, *rule lim-Ima''m*; *simp add*: *that*)

end

end