Concentrated Liquidity Market Making Operations

Mnacho Echenim

November 9, 2026

Abstract

Automated Market Makers (AMMs) are one of the cornerstones of decentralized finance. They enable users to exchange tokens without the need for order books as would be the case in traditional finance. They involve *liquidity providers*, whose tokens, usually called the *quote* and *base* tokens, can be used in the swap process in exchange for a fee, and *liquidity takers* who swap their tokens. The rules specifying the quantities of tokens that can swapped and those that act as fees are predefined and lead to several categories of AMMs.

Uniswap v3 introduced a new market-making design that improves capital efficiency by allowing liquidity providers to allocate their assets within selected price intervals. By concentrating liquidity over narrower ranges, providers may earn higher fee income than in earlier AMMs, where liquidity is generally distributed uniformly across all prices. Owing to its success, this design was adopted by several decentralized exchanges on various blockchains, including Trader Joe, PancakeSwap v3, Sunswap v3, and Sushiswap v3. These protocols are collectively referred to as *Concentrated Liquidity Market Makers* (CLMMs). Despite differences in implementation details, such as fee structures, tick spacing, or incentive mechanisms, they all rely on the same underlying principles.

In practice, liquidity takers can thus interact with multiple CLMM pools involving the same pair of tokens but different liquidity profiles or fee structures. A crucial task for them is to understand how these pools can be combined, both conceptually and computationally.

Based on the work in [1], we formalize several notions related to CLMMs, and introduce several operations on such pools that permit to derive an optimality result: if two pools admit the same fees, then the defined transformations permit to determine the optimal quantities of quote tokens to trade in each pool in order to recover as many base tokens as possible.

Contents

1 Preliminary definitions and results **3**

1.1 Misc 3

1.2 Support of a discrete function 8

2 Grid information **11**
2.1 Definitions . . . 11
2.2 Gross and net token quantities . . . 13
2.2.1 General definitions . . . 13
2.2.2 Finite support restriction . . . 14
2.3 Gross and net quantities of quote tokens . . . 15
2.3.1 Generic functions for quote tokens . . . 15
2.3.2 Finite support restriction . . . 16
2.4 Gross quote token quantity into a pool . . . 17
2.4.1 Function specialization . . . 17
2.4.2 Restriction to pools with a finite liquidity . . . 18
2.5 Net quote token quantity in a pool . . . 21
2.5.1 Function specialization . . . 21
2.5.2 Restriction to pools with a finite liquidity . . . 21
2.6 Gross and net quantities of base tokens . . . 22
2.6.1 Generic functions for base tokens . . . 22
2.6.2 Finite support restriction . . . 24
2.7 Gross base token quantity in a pool . . . 25
2.7.1 Function specialization . . . 25
2.7.2 Restriction to pools with a finite liquidity . . . 26
2.8 Net base token quantity in a pool . . . 27
2.8.1 Function specialization . . . 27
2.8.2 Restriction to pools with a finite liquidity . . . 28
2.9 Swapping tokens, market depth and slippage . . . 28
2.10 Identical profiles . . . 29

3 Grid refinement **31**
3.1 Encompassement properties . . . 31
3.2 Finer price grids . . . 32
3.3 Pools with finer grids and coinciding profiles . . . 38
3.4 Spanning grids . . . 41
3.5 Spanning grids and finite liquidity . . . 43

4 CLMM description **44**
4.1 Preliminary results . . . 44
4.2 Quote token addition and withdrawal in a CLMM . . . 49
4.3 Base token addition and withdrawal in a CLMM . . . 58
4.4 Market depth and slippage for finer CLMMs . . . 65
4.4.1 Finer pools . . . 65
4.4.2 Finer CLMMs with nonzero liquidity . . . 66

5 Inequalities related to fees **67**

6 CLMM transformations **73**
6.1 CLMM pool refinement 73
6.2 CLMM pool restriction and slice 81
6.3 CLMM pool join 87
6.4 CLMM pool combination 92
6.5 Optimality result on quote tokens 101
theory *CLMM-Misc* **imports** *HOL−Analysis.Analysis*

begin

1 Preliminary definitions and results

1.1 Misc

lemma *diff-min-le*:
assumes $(a::real) \leq b$
and $x \leq y$
shows $min\ x\ b - min\ x\ a \leq min\ y\ b - min\ y\ a$
⟨*proof*⟩

lemma *sum-ex-strict-pos*:
fixes $f\ g :: 'i \Rightarrow 'a::ordered\text{-}cancel\text{-}comm\text{-}monoid\text{-}add$
assumes *finite A*
and $\forall x{\in}A.\ 0 \leq f\ x$
and $\exists a{\in}A.\ 0 < f\ a$
shows $0 < sum\ f\ A$
⟨*proof*⟩

lemma *diff-inv-max-le*:
assumes $0 < a$
and $(a::real) \leq b$
and $x \leq y$
shows $inverse\ (max\ y\ a) - inverse\ (max\ y\ b) \leq$
$inverse\ (max\ x\ a) - inverse\ (max\ x\ b)$
⟨*proof*⟩

lemma *int-interval-insert*:
fixes $a::int$
assumes $a \leq b$
shows $\{a..<(b+1)\} = insert\ b\ \{a..<b\}$
⟨*proof*⟩

lemma *int-telescoping-sum*:
fixes $f::int \Rightarrow 'a::ab\text{-}group\text{-}add$

assumes $a \leq b$
shows $(\sum i \in \{a..<b\}.\ (f\ i - f\ (i+1))) = f\ a - (f\ b)$ ⟨*proof*⟩

lemma *int-telescoping-sum'*:
fixes f::*int* ⇒ '*a*::*ab-group-add*
assumes $a \leq b$
shows $(\sum i \in \{a..<b\}.\ (f\ (i+1) - f\ i)) = f\ b - (f\ a)$
⟨*proof*⟩

lemma *int-telescoping-sum-le'*:
fixes f::*int* ⇒ '*a*::*ab-group-add*
assumes $a \leq b$
shows $(\sum i \in \{a..b\}.\ (f\ (i+1) - f\ i)) = f\ (b+1) - (f\ a)$
⟨*proof*⟩

lemma *diff-sum-dcomp*:
fixes f::'*a* ⇒ *real*
assumes *finite A*
and $A = B \cup C$
and $B \cap C = \{\}$
shows $x + sum\ f\ A - (y + sum\ f\ B) = x + sum\ f\ C - y$
⟨*proof*⟩

lemma *sum-remove-el*:
assumes *finite A*
and $x \in A$
and $B = A - \{x\}$
shows $sum\ f\ A = f\ x + sum\ f\ B$
⟨*proof*⟩

lemma *int-set-bdd-above*:
fixes X::*int set*
assumes $X \neq \{\}$
and *bdd-above X*
shows $Sup\ X \in X\ \forall x \in X.\ x \leq Sup\ X$
⟨*proof*⟩

definition *wedge* **where**
$wedge\ f\ (i::int)\ sqp = (\lambda n.\ if\ n \leq i\ then\ f\ n\ else\ f\ (n-1))(i+1:=sqp)$

lemma *wedge-arg-lt*[*simp*]:
assumes $n \leq i$
shows $wedge\ f\ i\ sqp\ n = f\ n$ ⟨*proof*⟩

lemma *wedge-arg-gt*[*simp*]:
assumes $i+1 < n$
shows $wedge\ f\ i\ sqp\ n = f\ (n-1)$ ⟨*proof*⟩

lemma *wedge-arg-eq*[*simp*]:

shows *wedge f i sqp (i+1) = sqp* ⟨*proof*⟩

lemma *wedge-strict-mono*:
assumes *strict-mono f*
and *f i < sqp*
and *sqp < f (i+1)*
and *g = wedge f i sqp*
shows *strict-mono g* ⟨*proof*⟩

lemma *wedge-gt*:
assumes ∀ *i*. *x < f i*
and *x < sqp*
shows ∀ *i*. *x < wedge f j sqp i*
⟨*proof*⟩

lemma *wedge-ge*:
assumes ∀ *i*. *x ≤ f i*
and *x ≤ sqp*
shows ∀ *i*. *x ≤ wedge f j sqp i*
⟨*proof*⟩

lemma *wedge-lt*:
assumes ∀ *i*. *f i < x*
and *sqp < x*
shows ∀ *i*. *wedge f j sqp i < x*
⟨*proof*⟩

lemma *wedge-le*:
assumes ∀ *i*. *f i ≤ x*
and *sqp ≤ x*
shows ∀ *i*. *wedge f j sqp i ≤ x*
⟨*proof*⟩

lemma *wedge-images*:
shows ∀ *j*. ∃ *k*. *f j = wedge f i sqp k*
⟨*proof*⟩

lemma *wedge-images′*:
assumes *A* = {*j*. *j ≤ i*}
and *B* = {*j*. *i+1 < j*}
shows *wedge f i sqp k* ∈ *f‘A* ∪ (*f‘*((λ*i*. *i−1*) *‘B*)) ∪ {*sqp*}
⟨*proof*⟩

lemma *wedge-as-ubound*:
assumes ∀ (*r*::*real*). ∃ (*i*::*int*). *r < f i*
shows ∀ *r*. ∃ *k*. *r < wedge f i sqp k* ⟨*proof*⟩

lemma *wedge-as-lbound*:
assumes ∀ (*r*::*real*) > *0*. ∃ (*i*::*int*). *f i < r*

shows $\forall\, r > 0.\ \exists\, k.\ wedge\ f\ i\ sqp\ k < r$ ⟨*proof*⟩

lemma *wedge-arg-prop*:
shows $\{j.\ P\ (wedge\ f\ i\ sqp\ j)\} \subseteq \{j.\ j \leq i \wedge P\ (f\ j)\} \cup$
$\{j.\ i{+}1 < j \wedge P\ (f\ (j{-}1))\} \cup \{i{+}1\}$
⟨*proof*⟩

definition *one-cpl* **where**
one-cpl phi = $(\lambda(i{::}int).\ (1{::}real) - (phi\ i))$

definition *gross-fct* **where**
gross-fct f phi = $(\lambda i.\ f\ i\ /\ (one\text{-}cpl\ phi\ i))$

lemma *gross-fct-sgn*:
assumes *phi i* < $(1{::}real)$
shows $((0{::}real) \leq f\ i) \longleftrightarrow (0 \leq gross\text{-}fct\ f\ phi\ i)$ ⟨*proof*⟩

lemma *gross-fct-nz-eq*:
assumes *phi i* ≠ $(1{::}real)$
shows *f i* = *0* ⟷ *gross-fct f phi i* = *0* ⟨*proof*⟩

lemma *gross-fct-cong*:
assumes *f a* = *f′ b*
and *phi a* = *phi′ b*
shows *gross-fct f phi a* = *gross-fct f′ phi′ b* ⟨*proof*⟩

lemma *gross-fct-zero-if*:
assumes *f a* = *0*
shows *gross-fct f phi a* = *0* ⟨*proof*⟩

definition *fee-union* **where**
fee-union $(l1{::}real)$ *l2 f1 f2* = *(l1∗f1∗(1−f2) + l2∗f2∗(1−f1))/*
(l1∗(1−f2) + l2∗(1−f1))

lemma *fee-union-pos*:
assumes *0* ≤ *l1*
and *0* ≤ *l2*
and *0* ≤ *f1*
and *0* ≤ *f2*
and *f1* < *1*
and *f2* < *1*
shows *0* ≤ *fee-union l1 l2 f1 f2* ⟨*proof*⟩

lemma *fee-union-lt-1*:
assumes *0* ≤ *l1*
and *0* ≤ *l2*
and *0* ≤ *f1*
and *0* ≤ *f2*
and *f1* < *1*

and *f2 < 1*
shows *fee-union l1 l2 f1 f2 < 1*
⟨*proof*⟩

lemma *diff-inv-le*:
assumes *0 < (x::real)*
and $x \leq y$
and $y \leq z$
shows $(y - x)/(z*z) \leq inverse\ x - inverse\ y$
⟨*proof*⟩

lemma *diff-inv-le'*:
assumes *0 < (x::real)*
and $x \leq y$
and $y \leq z$
and $0 \leq a$
shows $a * (y - x)/(z*z) \leq a * (inverse\ x - inverse\ y)$
⟨*proof*⟩

lemma *diff-inv-sum-le'*:
assumes $\forall\, i \in I.\ (0::real) < f\ i$
and $\forall\, i \in I.\ f\ i \leq f\ (i+1)$
and $\forall\, i \in I.\ f\ (i+1) \leq z$
and $\forall\, i \in I.\ 0 \leq g\ i$
shows $sum\ (\lambda i.\ g\ i * (f\ (i+1) - f\ i))\ I\ /\ (z * z) \leq$
$sum\ (\lambda i.\ g\ i * (inverse\ (f\ i) - inverse\ (f\ (i+1))))\ I$
⟨*proof*⟩

lemma *diff-inv-ge*:
assumes *0 < (x::real)*
and $x \leq y$
and $y \leq z$
shows $inverse\ y - inverse\ z \leq (z - y)/(x*x)$
⟨*proof*⟩

lemma *diff-inv-ge'*:
assumes *0 < (x::real)*
and $x \leq y$
and $y \leq z$
and $0 \leq a$
shows $a * (inverse\ y - inverse\ z) \leq a * (z - y)/(x*x)$
⟨*proof*⟩

lemma *diff-inv-sum-ge'*:
assumes *(0::real) < z*
and $\forall\, i \in I.\ f\ i \leq f\ (i+1)$
and $\forall\, i \in I.\ z \leq f\ i$
and $\forall\, i \in I.\ 0 \leq g\ i$
shows $sum\ (\lambda i.\ g\ i * (inverse\ (f\ i) - inverse\ (f\ (i+1))))\ I \leq$

sum ($\lambda i.\ g\ i * (f\ (i+1) - f\ i))\ I\ /\ (z * z)$
⟨*proof*⟩

1.2 Support of a discrete function

definition *nz-support* **where**
nz-support f = $\{i.\ f\ i \neq 0\}$

lemma *nz-supportD*:
assumes *i*∈ *nz-support f*
shows *f i* ≠ *0* ⟨*proof*⟩

lemma *wedge-finite-nz-support*:
assumes *finite* (*nz-support f*)
shows *finite* (*nz-support* (*wedge f i sqp*))
⟨*proof*⟩

lemma *gross-nz-support-eq*:
assumes ∀ *i* ∈ *nz-support f*. *phi i* ≠ *1*
shows *nz-support f* = *nz-support* (*gross-fct f phi*)
⟨*proof*⟩

definition *idx-min* **where**
idx-min f = *Inf* (*nz-support f*)

definition *idx-max* **where**
idx-max f = *Sup* (*nz-support f*)

lemma *idx-max-bdd-above-ge*:
fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
assumes *f i* ≠ *0*
and *bdd-above* (*nz-support f*)
shows *i* ≤ *idx-max f*
⟨*proof*⟩

lemma *idx-min-bdd-below-le*:
fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
assumes *f i* ≠ *0*
and *bdd-below* (*nz-support f*)
shows *idx-min f* ≤ *i*
⟨*proof*⟩

lemma *idx-max-finite-ge*:
fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
assumes *f i* ≠ *0*
and *finite* (*nz-support f*)
shows *i* ≤ *idx-max f* ⟨*proof*⟩

lemma *idx-min-finite-le*:

fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $f\ i \neq 0$
and $finite\ (nz\text{-}support\ f)$
shows $idx\text{-}min\ f \leq i\ \langle proof \rangle$

lemma *idx-max-finite*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $nz\text{-}support\ f \neq \{\}$
and $finite\ (nz\text{-}support\ f)$
shows $idx\text{-}max\ f = Max\ (nz\text{-}support\ f)\ \langle proof \rangle$

lemma *idx-min-finite*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $nz\text{-}support\ f \neq \{\}$
and $finite\ (nz\text{-}support\ f)$
shows $idx\text{-}min\ f = Min\ (nz\text{-}support\ f)\ \langle proof \rangle$

lemma *idx-max-finite-in*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $nz\text{-}support\ f \neq \{\}$
and $finite\ (nz\text{-}support\ f)$
shows $f\ (idx\text{-}max\ f) \neq 0\ \langle proof \rangle$

lemma *idx-min-finite-in*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $nz\text{-}support\ f \neq \{\}$
and $finite\ (nz\text{-}support\ f)$
shows $f\ (idx\text{-}min\ f) \neq 0\ \langle proof \rangle$

lemma *idx-max-finite-gt*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $finite\ (nz\text{-}support\ f)$
and $idx\text{-}max\ f < i$
shows $f\ i = 0$
$\langle proof \rangle$

lemma *idx-min-finite-lt*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $finite\ (nz\text{-}support\ f)$
and $i < idx\text{-}min\ f$
shows $f\ i = 0$
$\langle proof \rangle$

lemma *idx-min-finite-max*:
fixes $f::'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::zero$
assumes $nz\text{-}support\ f \neq \{\}$
and $finite\ (nz\text{-}support\ f)$
and $\bigwedge j.\ j < i \Longrightarrow f\ j = 0$
shows $i \leq idx\text{-}min\ f$

⟨*proof*⟩

lemma *idx-min-max-finite*:
  fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
  assumes *nz-support f* ≠ {}
  and *finite* (*nz-support f*)
shows *idx-min f*≤ *idx-max f*
⟨*proof*⟩

lemma *idx-min-finiteI*:
  fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
  assumes *finite* (*nz-support f*)
  and *f i* ≠ *0*
  and ⋀*j*. *j* < *i*⟹ *f j* = *0*
shows *i* = *idx-min f*
⟨*proof*⟩

lemma *idx-min-finite-ge*:
  fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
  assumes *finite* (*nz-support f*)
  and *nz-support f* ≠ {}
  and ⋀*j*. *j* ≤ *i*⟹ *f j* = *0*
shows *i* ≤ *idx-min f*
  ⟨*proof*⟩

lemma *idx-max-finiteI*:
  fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
  assumes *finite* (*nz-support f*)
  and *f i* ≠ *0*
  and ⋀*j*. *j* > *i* ⟹ *f j* = *0*
shows *i* = *idx-max f*
⟨*proof*⟩

lemma *idx-max-finite-le*:
  fixes *f*::*'a*::*conditionally-complete-linorder* ⇒ *'b*::*zero*
  assumes *finite* (*nz-support f*)
  and *nz-support f* ≠ {}
  and ⋀*j*. *i*≤ *j* ⟹ *f j* = *0*
shows *idx-max f* ≤ *i*
  ⟨*proof*⟩

definition *idx-min-img* **where**
idx-min-img g f = *g* (*idx-min f*)

lemma *idx-min-img-eq*:
  assumes ∀ *x*. *f x* = *0* ⟷ *f' x* = *0*
  shows *idx-min-img g f* = *idx-min-img g f'* ⟨*proof*⟩

definition *idx-max-img* **where**

idx-max-img g f = g (idx-max f + 1)

lemma *idx-max-img-eq*:
assumes $\forall x.\ f\ x = 0 \longleftrightarrow f'\ x = 0$
shows *idx-max-img g f = idx-max-img g f' ⟨proof⟩*

lemma *non-zero-liq-interv*:
fixes *i*::*'a*::*conditionally-complete-linorder*
assumes *finite* (*nz-support L*)
and $L\ i \neq 0$
shows $i \in \{idx\text{-}min\ L\ ..\ idx\text{-}max\ L\}$
⟨*proof*⟩

end
theory *Grid-Information* **imports** *CLMM-Misc*

begin

2 Grid information

2.1 Definitions

A grid information consists of three functions defining the way a grid is associated to (square root) prices, the liquidity on each price range and the fees on each price range.

type-synonym *grid-info* = (*int*⇒ *real*) × (*int*⇒ *real*) × (*int*⇒*real*)

definition *grd*::*grid-info* ⇒ (*int*⇒ *real*) **where**
grd P = fst P

definition *lq*::*grid-info* ⇒ (*int*⇒ *real*) **where**
lq P = fst (*snd P*)

definition *fee*::*grid-info* ⇒ (*int*⇒ *real*) **where**
fee P = snd (*snd P*)

Although several results are formalized in a generalized setting, the pools of interest are those admitting a finite range with nonzero liquidity.

definition *finite-liq* **where**
finite-liq P ⟷ *finite* (*nz-support* (*lq P*))

lemma *finite-liqI*[*intro*]:
assumes *finite* $\{i.\ lq\ P\ i\ \neq 0\}$

shows *finite-liq P* ⟨*proof*⟩

lemma *finite-liqD*:
assumes *finite-liq P*
shows *finite* {*i*. *lq P i* ≠ *0*} ⟨*proof*⟩

definition *grd-min* **where**
grd-min P = *idx-min-img* (*grd P*) (*lq P*)

definition *grd-max* **where**
grd-max P = *idx-max-img* (*grd P*) (*lq P*)

lemma *grd-min-pos*:
assumes *nz-support* (*lq P*) ≠ {}
and ⋀*i*. *0* < *grd P i*
shows *0* < *grd-min P*
⟨*proof*⟩

lemma *grd-max-gt*:
assumes *nz-support* (*lq P*) ≠ {}
and ⋀*i*. *0* < *grd P i*
shows *0* < *grd-max P*
⟨*proof*⟩

locale *finite-nz-support* =
fixes *L*::*int* ⇒ *real*
assumes *fin-nz-sup*: *finite* (*nz-support L*)

locale *finite-liq-pool* =
fixes *P*
assumes *fin-liq*: *finite-liq P*

sublocale *finite-liq-pool* ⊆ *finite-nz-support lq P*
⟨*proof*⟩

context *finite-liq-pool*
begin

lemma *idx-max-mem*:
assumes *nz-support* (*lq P*) ≠ {}
shows *idx-max* (*lq P*) ∈ *nz-support* (*lq P*)
⟨*proof*⟩

lemma *idx-min-mem*:
assumes *nz-support* (*lq P*) ≠ {}
shows *idx-min* (*lq P*) ∈ *nz-support* (*lq P*)
⟨*proof*⟩

lemma *grd-min-max*:

assumes *nz-support* (*lq P*) ≠ {}
and *mono* (*grd P*)
shows *grd-min P* ≤ *grd-max P*
⟨*proof*⟩

lemma *finite-liq-gross-fct*:
shows *finite* {*i*. *gross-fct* (*lq P*) (*fee P*) *i* ≠ *0*}
⟨*proof*⟩

end

2.2 Gross and net token quantities

2.2.1 General definitions

We define generic functions that are afterwards instantiated to represent the gross (resp. net) quantities of base (resp. quote) tokens in a pool.

definition *rng-token* **where**
rng-token = (λ*dff L* (*pi*::*real*) *i*. ((*L i*)::*real*) ∗ (*dff pi* (*i*::*int*)))

lemma *rng-token-pos*:
assumes *0* ≤ *L i*
and *0* ≤ *dff x i*
shows *0* ≤ *rng-token dff L x i* ⟨*proof*⟩

lemma *rng-token-continuous-on*:
assumes *continuous-on A* (λ*pi*. *dff pi i*)
shows *continuous-on A* (λ*pi*. *rng-token dff L pi i*) ⟨*proof*⟩

(Anti)-monotonicity is preserved by the generic function *rng-token*.

lemma *rng-token-mono*:
assumes *0* ≤ *L i*
and *mono* (λ*pi*. *dff pi i*)
shows *mono* (λ*pi*. *rng-token dff L pi i*)
⟨*proof*⟩

lemma *rng-token-strict-mono*:
assumes (*0*::*real*) < *L i*
and *strict-mono* (λ*pi*. *dff pi i*)
shows *strict-mono* (λ*pi*. *rng-token dff L pi i*)
⟨*proof*⟩

lemma *rng-token-antimono*:
assumes *0* ≤ *L i*
and *antimono* (λ*pi*. *dff pi i*)
shows *antimono* (λ*pi*. *rng-token dff L pi i*)
⟨*proof*⟩

lemma *rng-token-add*:

assumes $\forall\, i.\ L\ i = L1\ i + L2\ i$
shows *rng-token dff L x i = rng-token dff L1 x i +*
rng-token dff L2 x i
⟨*proof*⟩

The generic function for the gross or net token quantities on the entire pool is obtained by summation of *rng-token* on all ranges.

definition *gen-token* **where**
gen-token = (λ*dff L pi*. (*infsum* (*rng-token dff L pi*) *UNIV*))

lemma *gen-token-pos*:
assumes $\forall\, i.\ 0 \leq L\ i$
and $\forall\, i.\ 0 \leq dff\ x\ i$
shows $0 \leq$ *gen-token dff L x* ⟨*proof*⟩

lemma *gen-token-mono*:
assumes $\forall\, i.\ 0 \leq L\ i$
and $\forall\, x.$ *rng-token dff L x summable-on UNIV*
and $\forall\, i.$ *mono* (λ*pi*. *dff pi i*)
shows *mono* (λ*pi*. *gen-token dff L pi*)
⟨*proof*⟩

lemma *gen-token-antimono*:
assumes $\forall\, i.\ 0 \leq L\ i$
and $\forall\, x.$ *rng-token dff L x summable-on UNIV*
and $\forall\, i.$ *antimono* (λ*pi*. *dff pi i*)
shows *antimono* (λ*pi*. *gen-token dff L pi*)
⟨*proof*⟩

2.2.2 Finite support restriction

context *finite-nz-support*

begin

lemma *finite-nonzero-summable*:
shows *rng-token dff L x summable-on UNIV*
⟨*proof*⟩

lemma *gen-token-antimono-finite*:
assumes $\forall\, i.\ 0 \leq L\ i$
and $\forall\, i.$ *antimono* (λ*pi*. *dff pi i*)
shows *antimono* (λ*pi*. *gen-token dff L pi*)
⟨*proof*⟩

lemma *gen-token-sum*:
shows *gen-token dff L x =*
sum (*rng-token dff L x*) {*i*. *L i* ≠ *0*}
⟨*proof*⟩

lemma *gen-token-continuous*:
assumes $\forall i.$ *continuous-on A* ($\lambda pi.$ *dff pi i*)
shows *continuous-on A* (*gen-token dff L*)
⟨*proof*⟩

lemma *gen-token-strict-mono*:
assumes $\forall i.$ $0 \leq L\ i$
and *nz-support L* $\neq \{\}$
and $\forall i.$ *strict-mono* ($\lambda pi.$ *dff pi i*)
shows *strict-mono* ($\lambda pi.$ *gen-token dff L pi*)
⟨*proof*⟩

lemma *gen-token-add*:
assumes $\forall i.$ $L\ i = L1\ i + L2\ i$
and $\forall i.$ $0 \leq L1\ i$
and $\forall i.$ $0 \leq L2\ i$
shows *gen-token dff L x* = *gen-token dff L1 x* + *gen-token dff L2 x*
⟨*proof*⟩

end

2.3 Gross and net quantities of quote tokens

2.3.1 Generic functions for quote tokens

definition *gamma-min-diff* **where**
gamma-min-diff gamma =
(λ(*pi*::*real*) *i*. (*min pi* (*gamma* (*i*+(*1*::*int*)))) − (*min pi* (*gamma i*)))

lemma *gamma-min-diff-pos*:
assumes *gamma i* ≤ *gamma* (*i*+*1*)
shows *0* ≤ *gamma-min-diff gamma x i*
⟨*proof*⟩

lemma *gamma-min-diff-continuous*:
shows *continuous-on A* (λ(*pi*::*real*). *gamma-min-diff gamma pi i*)
⟨*proof*⟩

lemma *gamma-min-diff-mono*:
fixes *gamma*::*int* ⇒ *real*
assumes *gamma i* ≤ *gamma* (*i*+*1*)
shows *mono* ($\lambda pi.$ *gamma-min-diff gamma pi i*)
⟨*proof*⟩

definition *rng-gen-quote* **where**
rng-gen-quote gamma = ($\lambda L\ pi\ i.$ *rng-token* (*gamma-min-diff gamma*) *L pi i*)

lemma *rng-gen-quote-pos*:
assumes *0* ≤ *L i*

and *gamma i* $\leq$ *gamma* (*i+1*)
shows *0* $\leq$ *rng-gen-quote gamma L x i* ⟨*proof*⟩

lemma *rng-gen-quote-continuous-on*:
shows *continuous-on A* (λ*pi*. *rng-gen-quote gamma L pi i*)
⟨*proof*⟩

lemma *rng-gen-quote-mono*:
assumes *0* $\leq$ *L i*
and *gamma i* $\leq$ *gamma* (*i+1*)
shows *mono* (λ*pi*. *rng-gen-quote gamma L pi i*)
⟨*proof*⟩

definition *gen-quote* **where**
gen-quote = (λ *gamma L pi*. *gen-token* (*gamma-min-diff gamma*) *L pi*)

lemma *gen-quote-zero*:
assumes *mono gamma*
and ⋀*i*. *gamma i* < *sqp* ⟹ *L i* = *0*
shows *gen-quote gamma L sqp* = *0* ⟨*proof*⟩

lemma *gen-quote-pos*:
assumes ∀ *i*. *0* $\leq$ *L i*
and ∀ *i*. *gamma i* $\leq$ *gamma* (*i+1*)
shows *0* $\leq$ *gen-quote gamma L x* ⟨*proof*⟩

lemma *gen-quote-mono*:
assumes ∀ *i*. *0* $\leq$ *L i*
and ∀ *x*. *rng-token* (*gamma-min-diff gamma*) *L x summable-on UNIV*
and ∀ *i*. *gamma i* $\leq$ *gamma* (*i+1*)
shows *mono* (*gen-quote gamma L*) ⟨*proof*⟩

2.3.2 Finite support restriction

context *finite-nz-support*
begin

lemma *gen-quote-mono-finite*:
assumes ∀ *i*. *0* $\leq$ *L i*
and ∀ *i*. *gamma i* $\leq$ *gamma* (*i+1*)
shows *mono* (*gen-quote gamma L*)
⟨*proof*⟩

lemma *gen-quote-continuous*:
shows *continuous-on A* (*gen-quote gamma L*) ⟨*proof*⟩

lemma *gen-quote-IVT*:
assumes (*idx-min-img gamma L*) $\leq$ (*idx-max-img gamma L*)
and *gen-quote gamma L* (*idx-min-img gamma L*) $\leq$ *y*

and $y \leq$ *gen-quote gamma L* (*idx-max-img gamma L*)
shows $\exists\, pi \geq$ (*idx-min-img gamma L*). $pi \leq$ *idx-max-img gamma L* $\wedge$
gen-quote gamma L pi $= y$
⟨*proof*⟩

lemma *gen-quote-ne*:
assumes (*idx-min-img gamma L*) $\leq$ (*idx-max-img gamma L*)
and *gen-quote gamma L* (*idx-min-img gamma L*) $\leq y$
and $y \leq$ *gen-quote gamma L* (*idx-max-img gamma L*)
shows (*gen-quote gamma L*)−‘ $\{y\} \neq \{\}$ ⟨*proof*⟩

lemma *finite-support-sum*:
assumes $\bigwedge i$. *L i* $= 0 \Longrightarrow$ *f L i* $= 0$
shows *infsum* (*rng-token dff* (*f L*) *x*) *UNIV* =
sum (*rng-token dff* (*f L*) *x*) $\{i.$ *L i* $\neq 0\}$
⟨*proof*⟩

lemma *gen-quote-plus*:
assumes $\forall\, i.$ *L i* $=$ *L1 i* $+$ *L2 i*
and $\forall\, i.$ $0 \leq$ *L1 i*
and $\forall\, i.$ $0 \leq$ *L2 i*
shows *gen-quote gam L x* = *gen-quote gam L1 x* + *gen-quote gam L2 x*
⟨*proof*⟩

end

2.4 Gross quote token quantity into a pool

2.4.1 Function specialization

When the quote tokens are added to a pool, fees are to be taken into account: if a user adds a quantity q of tokens into a pool, the computation of the amount of base tokens received is based in $q \cdot (1 - \varphi)$.

definition *rng-quote-gross* **where**
rng-quote-gross P = *rng-gen-quote* (*grd P*) (*gross-fct* (*lq P*) (*fee P*))

lemma *rng-quote-gross-pos*:
assumes $0 \leq$ *gross-fct* (*lq P*) (*fee P*) *i*
and *grd P i* $\leq$ *grd P* (*i*+*1*)
shows $0 \leq$ *rng-quote-gross P pi i* ⟨*proof*⟩

lemma *rng-quote-gross-continuous-on*:
shows *continuous-on A* (λ*pi*. *rng-quote-gross P pi i*)
⟨*proof*⟩

lemma *rng-quote-gross-mono*:
assumes $0 \leq$ *gross-fct* (*lq P*) (*fee P*) *i*
and *grd P i* $\leq$ *grd P* (*i*+*1*)
shows *mono* (λ*pi*. *rng-quote-gross P pi i*) ⟨*proof*⟩

definition *quote-gross* **where**
quote-gross P = gen-quote (grd P) (gross-fct (lq P) (fee P))

lemma *quote-gross-pos*:
assumes $\forall i.\ 0 \leq$ *gross-fct (lq P) (fee P) i*
and $\forall i.$ *grd P i* $\leq$ *grd P (i+1)*
shows *0* $\leq$ *quote-gross P x* ⟨*proof*⟩

lemma *quote-gross-mono*:
assumes $\forall i.\ 0 \leq$ *(lq P) i*
and $\forall i.$ *(fee P) i* $<$ *1*
and $\forall i.$ *(grd P) i* $\leq$ *(grd P) (i+1)*
and $\forall x.$ *rng-token (gamma-min-diff (grd P)) (gross-fct (lq P) (fee P)) x*
summable-on UNIV
shows *mono (quote-gross P)* ⟨*proof*⟩

lemma *gen-quote-grd-min*:
assumes *mono (grd P)*
and *finite (nz-support L)*
and *nz-support L* $\neq \{\}$
and *nz-support L = nz-support (lq P)*
shows *gen-quote (grd P) L (grd-min P) = 0*
⟨*proof*⟩

Definition of the grid point that is reached in a pool for a given gross quantity of quote tokens.

definition *quote-reach* **where**
quote-reach = $(\lambda P\ y.$
if y = 0 then (grd-min P)
else Inf $((quote\text{-}gross\ P) - `\ \{y\})$)

2.4.2 Restriction to pools with a finite liquidity

context *finite-liq-pool*

begin

lemma *quote-gross-mono-finite*:
assumes $\forall i.\ 0 \leq$ *(lq P) i*
and $\forall i.$ *(fee P) i* $<$ *1*
and $\forall i.$ *(grd P) i* $\leq$ *(grd P) (i+1)*
shows *mono (quote-gross P)*
⟨*proof*⟩

lemma *quote-gross-mono-finite′*:
assumes $\forall i.\ 0 \leq$ *(lq P) i*
and $\forall i.$ *(fee P) i* $<$ *1*
and *mono (grd P)*

shows *mono* (*quote-gross P*)
⟨*proof*⟩

lemma *quote-gross-continuous*:
shows *continuous-on A* (*quote-gross P*) ⟨*proof*⟩

lemma *quote-gross-IVT*:
assumes ∀ *i*. *fee P i* ≠ *1*
and *grd-min P* ≤ *grd-max P*
and *quote-gross P* (*grd-min P*) ≤ *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
shows ∃ *pi* ≥ (*grd-min P*). *pi* ≤ (*grd-max P*) ∧
quote-gross P pi = *y*
⟨*proof*⟩

lemma *quote-gross-ne*:
assumes ∀ *i*. *fee P i* ≠ *1*
and *grd-min P* ≤ *grd-max P*
and *quote-gross P* (*grd-min P*) ≤ *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
shows *quote-gross P*−‘ {*y*} ≠ {} ⟨*proof*⟩

lemma *quote-gross-grd-min*:
assumes *mono* (*grd P*)
shows *quote-gross P* (*grd-min P*) = *0*
⟨*proof*⟩

lemma *quote-reach-mem*:
assumes ∀ *i*. *0* ≤ *lq P i*
and ∀ *i*. *fee P i* < *1*
and *mono* (*grd P*)
and *0* ≤ *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
shows *quote-reach P y* ∈ *quote-gross P*−‘ {*y*}
⟨*proof*⟩

lemma *quote-gross-inv-strict-mono*:
assumes *mono* (*quote-gross P*)
and *quote-gross P sqp′* < *y*
and *sqp* ∈ *quote-gross P* −‘ {*y*}
shows *sqp′* < *sqp*
⟨*proof*⟩

lemma *quote-gross-inv-bounded*:
assumes *mono* (*quote-gross P*)
and *quote-gross P* (*grd-min P*) < *y*
and *y* < *quote-gross P* (*grd-max P*)
shows ∀ *sqp′* ∈ *quote-gross P* −‘ {*y*}.
dist (*grd-min P*) *sqp′* ≤ *grd-max P* − *grd-min P*

⟨*proof*⟩

lemma *quote-gross-bdd-below*:
assumes *mono* (*quote-gross P*)
and *quote-gross P* (*grd-min P*) < *y*
shows *bdd-below* (*quote-gross P* − ‘{*y*}) ⟨*proof*⟩

lemma *quote-reach-le*:
assumes ∀ *i*. *0* ≤ *lq P i*
and ∀ *i*. *fee P i* < *1*
and *mono* (*grd P*)
and *0* < *y*
and *sqp* ∈ *quote-gross P* − ‘{*y*}
shows *quote-reach P y* ≤ *sqp*
⟨*proof*⟩

lemma *quote-reach-gross-le*:
assumes ∀ *i*. *0* ≤ *lq P i*
and ∀ *i*. *fee P i* < *1*
and *mono* (*grd P*)
and *grd-min P* ≤ *sqp*
shows *quote-reach P* (*quote-gross P sqp*) ≤ *sqp*
⟨*proof*⟩

lemma *quote-gross-reach-eq*:
assumes ∀ *i*. *0* ≤ *lq P i*
and ∀ *i*. *fee P i* < *1*
and *mono* (*grd P*)
and *0* ≤ *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
shows *quote-gross P* (*quote-reach P y*) = *y*
⟨*proof*⟩

lemma *quote-reach-ge*:
assumes ∀ *i*. *0* ≤ *lq P i*
and ∀ *i*. *fee P i* < *1*
and *mono* (*grd P*)
and *grd-min P* ≤ *grd-max P*
and *0* < *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
shows *grd-min P* ≤ *quote-reach P y*
⟨*proof*⟩

end

2.5 Net quote token quantity in a pool

2.5.1 Function specialization

There are no fees to take into account when tokens are withdrawn from a pool.

definition *rng-quote-net* **where**
rng-quote-net P = rng-gen-quote (grd P) (lq P)

lemma *rng-quote-net-pos*:
assumes *0 ≤ (lq P) i*
and *grd P i ≤ grd P (i+1)*
shows *0 ≤ rng-quote-net P x i ⟨proof⟩*

lemma *rng-quote-net-continuous-on*:
shows *continuous-on A (λpi. rng-quote-net P pi i)*
⟨proof⟩

lemma *rng-quote-net-mono*:
assumes *0 ≤ (lq P) i*
and *grd P i ≤ grd P (i+1)*
shows *mono (λpi. rng-quote-net P pi i) ⟨proof⟩*

definition *quote-net* **where**
quote-net P = gen-quote (grd P) (lq P)

lemma *quote-net-pos*:
assumes *∀ i. 0 ≤ (lq P) i*
and *∀ i. grd P i ≤ grd P (i+1)*
shows *0 ≤ quote-net P x ⟨proof⟩*

lemma *quote-net-mono*:
assumes *∀ i. 0 ≤ (lq P) i*
and *∀ i. (fee P) i < 1*
and *∀ i. (grd P) i ≤ (grd P) (i+1)*
and *∀ x. rng-token (gamma-min-diff (grd P)) (lq P) x summable-on UNIV*
shows *mono (quote-net P) ⟨proof⟩*

2.5.2 Restriction to pools with a finite liquidity

context *finite-liq-pool*
begin

lemma *quote-net-continuous*:
shows *continuous-on A (quote-net P) ⟨proof⟩*

lemma *quote-net-IVT*:
assumes *∀ i. fee P i ≠ 1*
and *grd-min P ≤ grd-max P*

and *quote-net P* (*grd-min P*) ≤ *y*
and *y* ≤ *quote-net P* (*grd-max P*)
shows ∃ *pi* ≥ (*grd-min P*). *pi* ≤ (*grd-max P*) ∧
quote-net P pi = *y*
⟨*proof*⟩

lemma *quote-net-ne*:
assumes ∀ *i*. *fee P i* ≠ *1*
and *grd-min P* ≤ *grd-max P*
and *quote-net P* (*grd-min P*) ≤ *y*
and *y* ≤ *quote-net P* (*grd-max P*)
shows *quote-net P* − ‘ {*y*} ≠ {} ⟨*proof*⟩

lemma *quote-net-mono-finite-liq*:
assumes ∀ *i*. *0* ≤ (*lq P*) *i*
and ∀ *i*. (*fee P*) *i* < *1*
and ∀ *i*. (*grd P*) *i* ≤ (*grd P*) (*i*+*1*)
shows *mono* (*quote-net P*) ⟨*proof*⟩

end

2.6 Gross and net quantities of base tokens

2.6.1 Generic functions for base tokens

definition *inv-gamma-max-diff* **where**
inv-gamma-max-diff = (λ*gamma* (*pi*::*real*) *i*. *inverse* (*max pi* (*gamma i*)) −
inverse (*max pi* (*gamma* (*i*+(*1*::*int*)))))

lemma *inv-max-pos*:
assumes *0* < *a*
and (*a*::*real*) ≤ *b*
shows *0* ≤ *inverse* (*max x a*) − *inverse* (*max x b*)
⟨*proof*⟩

lemma *inv-gamma-max-diff-pos*:
assumes *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and *0* < *gamma i*
shows *0* ≤ *inv-gamma-max-diff gamma x i* ⟨*proof*⟩

lemma *inv-gamma-max-diff-continuous*:
assumes *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and *0* < *gamma i*
shows *continuous-on A* (λ*pi*. *inv-gamma-max-diff gamma pi i*)
⟨*proof*⟩

lemma *inv-gamma-max-diff-antimono*:
assumes *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and *0* < *gamma i*
shows *antimono* (λ*pi*. *inv-gamma-max-diff gamma pi i*)

⟨*proof*⟩

definition *rng-gen-base* **where**
rng-gen-base =
(λ*gamma L pi i*. *rng-token* (*inv-gamma-max-diff gamma*) *L pi i*)

lemma *rng-gen-base-pos*:
assumes *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and *0* < *gamma i*
and *0* ≤ *L i*
shows *0* ≤ *rng-gen-base gamma L x i* ⟨*proof*⟩

lemma *rng-gen-base-continuous-on*:
assumes *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and *0* < *gamma i*
shows *continuous-on A* (λ*pi*. *rng-gen-base gamma L pi i*) ⟨*proof*⟩

lemma *rng-gen-base-antimono*:
assumes *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and *0* < *gamma i*
and *0* ≤ *L i*
shows *antimono* (λ*pi*. *rng-gen-base gamma L pi i*)
⟨*proof*⟩

definition *gen-base* **where**
gen-base = (λ*gamma L pi*. *gen-token* (*inv-gamma-max-diff gamma*) *L pi*)

lemma *gen-base-pos*:
assumes ∀ *i*. *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and ∀ *i*. *0* < *gamma i*
and ∀ *i*. *0* ≤ *L i*
shows *0* ≤ *gen-base gamma L x* ⟨*proof*⟩

lemma *gen-base-antimono*:
assumes ∀ *x*. *rng-token* (*inv-gamma-max-diff gamma*) *L x summable-on UNIV*
and ∀ *i*. *gamma i* ≤ *gamma* (*i* +(*1*::*int*))
and ∀ *i*. *0* < *gamma i*
and ∀ *i*. *0* ≤ *L i*
shows *antimono* (*gen-base gamma L*) ⟨*proof*⟩

lemma *gen-base-zero*:
assumes *mono gamma*
and ⋀*i*. *sqp* < *gamma* (*i*+*1*) ⟹ *L i* = *0*
shows *gen-base gamma L sqp* = *0* ⟨*proof*⟩

lemma *gen-base-grd-max*:
assumes *mono* (*grd P*)
and *finite* (*nz-support L*)
and *nz-support L* ≠ {}

and *nz-support L = nz-support (lq P)*
shows *gen-base (grd P) L (grd-max P) = 0*
⟨*proof*⟩

2.6.2 Finite support restriction

context *finite-nz-support*
begin

lemma *gen-base-continuous*:
assumes ∀ *i. gamma i ≤ gamma (i +(1::int))*
and ∀ *i. 0 < gamma i*
shows *continuous-on A (gen-base gamma L)* ⟨*proof*⟩

lemma *gen-base-IVT*:
assumes ∀ *i. gamma i ≤ gamma (i +(1::int))*
and ∀ *i. 0 < gamma i*
and *(idx-min-img gamma L) ≤ (idx-max-img gamma L)*
and *gen-base gamma L (idx-max-img gamma L) ≤ y*
and *y ≤ gen-base gamma L (idx-min-img gamma L)*
shows ∃ *pi ≥ (idx-min-img gamma L). pi ≤ (idx-max-img gamma L) ∧*
gen-base gamma L pi = y
⟨*proof*⟩

lemma *gen-base-ne*:
assumes ∀ *i. gamma i ≤ gamma (i +(1::int))*
and ∀ *i. 0 < gamma i*
and *(idx-min-img gamma L) ≤ (idx-max-img gamma L)*
and *gen-base gamma L (idx-max-img gamma L) ≤ y*
and *y ≤ gen-base gamma L (idx-min-img gamma L)*
shows *(gen-base gamma L)−‘ {y} ≠ {}* ⟨*proof*⟩

lemma *gen-base-antimono-finite*:
assumes ∀ *i. gamma i ≤ gamma (i +(1::int))*
and ∀ *i. 0 < gamma i*
and ∀ *i. 0 ≤ L i*
shows *antimono (gen-base gamma L)*
⟨*proof*⟩

lemma *gen-base-gross*:
assumes ∀ *i. L i = L1 i + L2 i*
and ∀ *i. 0 ≤ L1 i*
and ∀ *i. 0 ≤ L2 i*
shows *gen-base gam L x = gen-base gam L1 x + gen-base gam L2 x*
⟨*proof*⟩

end

2.7 Gross base token quantity in a pool

2.7.1 Function specialization

definition *rng-base-gross* **where**
rng-base-gross P = rng-gen-base (grd P) (gross-fct (lq P) (fee P))

lemma *rng-base-gross-pos*:
assumes *0 ≤ gross-fct (lq P) (fee P) i*
and *grd P i ≤ grd P (i+1)*
and *0 < grd P i*
shows *0 ≤ rng-base-gross P x i ⟨proof⟩*

lemma *rng-base-gross-continuous-on*:
assumes *grd P i ≤ grd P (i+1)*
and *0 < grd P i*
shows *continuous-on A (λpi. rng-base-gross P pi i)*
⟨proof⟩

lemma *rng-base-gross-mono*:
assumes *0 ≤ gross-fct (lq P) (fee P) i*
and *grd P i ≤ grd P (i+1)*
and *0 < grd P i*
shows *antimono (λpi. rng-base-gross P pi i) ⟨proof⟩*

definition *base-gross* **where**
base-gross P = gen-base (grd P) (gross-fct (lq P) (fee P))

lemma *base-gross-pos*:
assumes *∀ i. 0 ≤ gross-fct (lq P) (fee P) i*
and *∀ i. grd P i ≤ grd P (i+1)*
and *∀ i. 0 < grd P i*
shows *0 ≤ base-gross P x ⟨proof⟩*

lemma *base-gross-antimono*:
assumes *∀ i. 0 ≤ (lq P) i*
and *∀ i. (fee P) i < 1*
and *∀ i. (grd P) i ≤ (grd P) (i+1)*
and *∀ i. 0 < grd P i*
and *∀ x. rng-token (inv-gamma-max-diff (grd P)) (gross-fct (lq P) (fee P)) x*
summable-on UNIV
shows *antimono (base-gross P) ⟨proof⟩*

lemma *base-gross-grd-max*:
assumes *mono (grd P)*
and *finite (nz-support (lq P))*
shows *base-gross P (grd-max P) = 0*
⟨proof⟩

definition *base-reach* **where**

base-reach = (λ*P y*.
  if y = *0*
  then (*grd-max P*)
  else Sup ((*base-gross P*)−‘ {*y*}))

2.7.2 Restriction to pools with a finite liquidity

context *finite-liq-pool*
begin

lemma *base-gross-continuous*:
  assumes ∀ *i*. *grd P i* ≤ *grd P* (*i*+*1*)
  and ∀ *i*. *0* < *grd P i*
shows *continuous-on A* (*base-gross P*) ⟨*proof*⟩

lemma *base-gross-IVT*:
  assumes ∀ *i*. *grd P i* ≤ *grd P* (*i*+*1*)
  and ∀ *i*. *0* < *grd P i*
  and ∀ *i*. *fee P i* ≠ *1*
  and *grd-min P* ≤ *grd-max P*
  and *base-gross P* (*grd-max P*) ≤ *y*
  and *y* ≤ *base-gross P* (*grd-min P*)
shows ∃ *pi* ≥ (*grd-min P*). *pi* ≤ (*grd-max P*) ∧
  base-gross P pi = *y*
⟨*proof*⟩

lemma *base-gross-ne*:
  assumes ∀ *i*. *grd P i* ≤ *grd P* (*i*+*1*)
  and ∀ *i*. *0* < *grd P i*
  and ∀ *i*. *fee P i* ≠ *1*
  and *grd-min P* ≤ *grd-max P*
  and *base-gross P* (*grd-max P*) ≤ *y*
  and *y* ≤ *base-gross P* (*grd-min P*)
shows *base-gross P*−‘ {*y*} ≠ {} ⟨*proof*⟩

lemma *base-gross-antimono-finite*:
  assumes ∀ *i*. *0* ≤ (*lq P*) *i*
  and ∀ *i*. *grd P i* ≤ *grd P* (*i*+*1*)
  and ∀ *i*. *0* < *grd P i*
  and ∀ *i*. (*fee P*) *i* < *1*
shows *antimono* (*base-gross P*) ⟨*proof*⟩

lemma *base-reach-mem*:
  assumes ∀ *i*. *grd P i* ≤ *grd P* (*i*+*1*)
  and ∀ *i*. *0* < *grd P i*
  and ∀ *i*. *fee P i* < *1*
  and ∀ *i*. *0* ≤ *lq P i*
  and *mono* (*grd P*)
  and *grd-min P* ≤ *grd-max P*

and $0 \leq y$
and $y \leq$ *base-gross* P (*grd-min* P)
shows *base-reach* P $y \in$ *base-gross* $P - `$ $\{y\}$
⟨*proof*⟩

lemma *base-gross-dwn*:
assumes $\forall i.$ *grd* P $i \leq$ *grd* P $(i+1)$
and $\forall i.$ $0 <$ *grd* P i
and $\forall i.$ *fee* P $i < 1$
and $\forall i.$ $0 \leq$ *lq* P i
and *mono* (*grd* P)
and *grd-min* $P \leq$ *grd-max* P
and $0 \leq y$
and $y \leq$ *base-gross* P (*grd-min* P)
shows *base-gross* P (*base-reach* P y) $= y$
⟨*proof*⟩

end

2.8 Net base token quantity in a pool

2.8.1 Function specialization

definition *rng-base-net* **where**
rng-base-net $P =$ *rng-gen-base* (*grd* P) (*lq* P)

lemma *rng-base-net-pos*:
assumes *grd* P $i \leq$ *grd* P $(i +(1::int))$
and $0 <$ *grd* P i
and $0 \leq$ *lq* P i
shows $0 \leq$ *rng-base-net* P x i ⟨*proof*⟩

lemma *rng-base-net-continuous-on*:
assumes *grd* P $i \leq$ *grd* P $(i +(1::int))$
and $0 <$ *grd* P i
shows *continuous-on* A $(\lambda pi.$ *rng-base-net* P *pi* $i)$
⟨*proof*⟩

lemma *rng-base-net-mono*:
assumes *grd* P $i \leq$ *grd* P $(i +(1::int))$
and $0 <$ *grd* P i
and $0 \leq$ *lq* P i
shows *antimono* $(\lambda pi.$ *rng-base-net* P *pi* $i)$ ⟨*proof*⟩

definition *base-net* **where**
base-net $P =$ *gen-base* (*grd* P) (*lq* P)

lemma *base-net-pos*:
assumes $\forall i.$ *grd* P $i \leq$ *grd* P $(i +(1::int))$
and $\forall i.$ $0 <$ *grd* P i

and $\forall\, i.\ 0 \leq lq\ P\ i$
shows $0 \leq$ *base-net* $P\ x\ \langle proof \rangle$

2.8.2 Restriction to pools with a finite liquidity

context *finite-liq-pool*
begin

lemma *base-net-continuous*:
assumes $\forall\, i.$ *grd* $P\ i \leq$ *grd* $P\ (i + (1::int))$
and $\forall\, i.\ 0 <$ *grd* $P\ i$
shows *continuous-on* A (*base-net* P) $\langle proof \rangle$

lemma *base-net-IVT*:
assumes $\forall\, i.$ *grd* $P\ i \leq$ *grd* $P\ (i + (1::int))$
and $\forall\, i.\ 0 <$ *grd* $P\ i$
and *grd-min* $P \leq$ *grd-max* P
and *base-net* P (*grd-max* P) $\leq y$
and $y \leq$ *base-net* P (*grd-min* P)
shows $\exists\, pi \geq$ (*grd-min* P). $pi \leq$ (*grd-max* P) $\wedge$
base-net $P\ pi = y$
$\langle proof \rangle$

lemma *base-net-ne*:
assumes $\forall\, i.$ *grd* $P\ i \leq$ *grd* $P\ (i + (1::int))$
and $\forall\, i.\ 0 <$ *grd* $P\ i$
and *grd-min* $P \leq$ *grd-max* P
and *base-net* P (*grd-max* P) $\leq y$
and $y \leq$ *base-net* P (*grd-min* P)
shows *base-net* $P -$ ‘ $\{y\} \neq \{\}$ $\langle proof \rangle$

lemma *base-net-antimono-finite*:
assumes $\forall\, i.$ *grd* $P\ i \leq$ *grd* $P\ (i + (1::int))$
and $\forall\, i.\ 0 <$ *grd* $P\ i$
and $\forall\, i.\ 0 \leq lq\ P\ i$
shows *antimono* (*base-net* P) $\langle proof \rangle$

end

2.9 Swapping tokens, market depth and slippage

Given a grid point π and a quantity y of quote tokens to add to the pool, this function computes the amount of base tokens that are retrieved from the pool.

definition *quote-swap* **where**
quote-swap $P = (\lambda pi\ y.$
base-net $P\ pi -$ *base-net* P (*quote-reach* P ($y +$ *quote-gross* $P\ pi$)))

Given a grid point π and a quantity x of base tokens to add to the pool,

this function computes the amount of quote tokens that are retrieved from the pool.

definition *base-swap* **where**
base-swap P = (λ*pi x*.
  quote-net P pi − *quote-net P* (*base-reach P* (*x* + *base-gross P pi*)))

The market depth in a pool takes as arguments two grid points π and π', and returns the amounts of base or quote tokens that have to be added to the pool for its state to get from π to π'.

definition *mkt-depth* **where**
mkt-depth P = (λ *pi pi'*. *if pi* < *pi' then* (*base-net P pi* − *base-net P pi'*)
  else (*quote-net P pi* − *quote-net P pi'*))

Base and quote slippages relate the amount of tokens withdrawn from the pool from those given by an infinitesimally small amount of tokens and that can be deduced from the grid point.

definition *quote-slippage* **where**
quote-slippage P = (λ*pi y*. *y*/(*quote-swap P pi y* ∗ *pi* ∗ *pi*) − *1*)

definition *base-slippage* **where**
base-slippage P = (λ*pi x*. *base-swap P pi x*/(*x* ∗ *pi* ∗ *pi*) − *1*)

2.10 Identical profiles

definition *id-grid-on* **where**
id-grid-on P P' I ⟷ (∀ *i*∈ *I*. *grd P i* = *grd P' i*)

lemma *id-grid-onI*[*intro*]:
  assumes ⋀*i*. *i*∈ *I* ⟹ *grd P i* = *grd P' i*
  shows *id-grid-on P P' I* ⟨*proof*⟩

lemma *id-grid-onD*[*dest*]:
  assumes *id-grid-on P P' I*
  and *i*∈ *I*
shows *grd P i* = *grd P' i* ⟨*proof*⟩

lemma *id-grid-on-comm*:
  assumes *id-grid-on P P' I*
  shows *id-grid-on P' P I*
  ⟨*proof*⟩

lemma *id-grid-on-mono*:
  assumes *id-grid-on P P' I*
  and *I'* ⊆ *I*
shows *id-grid-on P P' I'* ⟨*proof*⟩

definition *same-nz-liq-on* **where**
same-nz-liq-on P P' I ⟷ *id-grid-on P P' I* ∧

$(\forall\, i \in I.\ (lq\ P\ i = 0) \longleftrightarrow (lq\ P'\ i = 0))$

lemma *same-nz-liq-onI*[*intro*]:
assumes *id-grid-on P P' I*
and $\bigwedge i.\ i \in I \Longrightarrow ((lq\ P\ i = 0) \longleftrightarrow (lq\ P'\ i = 0))$
shows *same-nz-liq-on P P' I* ⟨*proof*⟩

lemma *same-nz-liq-onD*[*dest*]:
assumes *same-nz-liq-on P P' I*
and $i \in I$
shows *grd P i* = *grd P' i* $(lq\ P\ i = 0) \longleftrightarrow (lq\ P'\ i = 0)$
⟨*proof*⟩

lemma *same-nz-liq-on-comm*:
assumes *same-nz-liq-on P P' I*
shows *same-nz-liq-on P' P I*
⟨*proof*⟩

lemma *same-nz-liq-on-mono*:
assumes *same-nz-liq-on P P' I*
and $I' \subseteq I$
shows *same-nz-liq-on P P' I'*
⟨*proof*⟩

definition *fee-diff-on* **where**
fee-diff-on P P' I $\longleftrightarrow$ *id-grid-on P P' I* $\wedge$ $(\forall\, i \in I.\ lq\ P\ i = lq\ P'\ i)$

lemma *fee-diff-onI*[*intro*]:
assumes *id-grid-on P P' I*
and $\bigwedge i.\ i \in I \Longrightarrow lq\ P\ i = lq\ P'\ i$
shows *fee-diff-on P P' I*
⟨*proof*⟩

lemma *fee-diff-onD*[*dest*]:
assumes *fee-diff-on P P' I*
shows *id-grid-on P P' I* $\forall\, i \in I.\ lq\ P\ i = lq\ P'\ i$
⟨*proof*⟩

lemma *fee-diff-on-nz-liq*:
assumes *fee-diff-on P P' I*
shows *same-nz-liq-on P P' I* ⟨*proof*⟩

lemma *fee-diff-on-comm*:
assumes *fee-diff-on P P' I*
shows *fee-diff-on P' P I*
⟨*proof*⟩

lemma *fee-diff-on-mono*:
assumes *fee-diff-on P P' I*

and $I' \subseteq I$
shows *fee-diff-on P P' I'*
⟨*proof*⟩

3 Grid refinement

We define the notion of pool refinement, that characterizes when a pool admits a finer price grid than another one but exhibits the same behavior.

3.1 Encompassement properties

definition *encomp-at* **where**
encomp-at gamma1 gamma2 i k ≡ *gamma2 k* ≤ *gamma1 i* ∧
gamma1 (*i*+*1*) ≤ *gamma2* (*k*+*1*)

lemma *encomp-atD1*:
assumes *encomp-at gamma1 gamma2 i k*
shows *gamma2 k* ≤ *gamma1 i*
⟨*proof*⟩

lemma *encomp-atD2*:
assumes *encomp-at gamma1 gamma2 i k*
shows *gamma1* (*i*+*1*) ≤ *gamma2* (*k*+*1*)
⟨*proof*⟩

lemma *encomp-atI*[*intro*]:
assumes *gamma2 k* ≤ *gamma1 i*
and *gamma1* (*i*+*1*) ≤ *gamma2* (*k*+*1*)
shows *encomp-at gamma1 gamma2 i k* ⟨*proof*⟩

definition *encompassed* **where**
encompassed gamma1 gamma2 k = {*i*::*int*. *encomp-at gamma1 gamma2 i k*}

lemma *encompassed-convex*:
assumes (*i*::*int*) ∈ *encompassed gamma1 gamma2 k*
and *j* ∈ *encompassed gamma1 gamma2 k*
and *i* ≤ *l*
and *l* ≤ *j*
and *mono gamma1*
shows *l* ∈ *encompassed gamma1 gamma2 k* ⟨*proof*⟩

lemma *encompassed-interval*:
assumes *mono gamma1*
and *finite* (*encompassed gamma1 gamma2 k*)
and *encompassed gamma1 gamma2 k* ≠ {}
shows *encompassed gamma1 gamma2 k* =
{*Min* (*encompassed gamma1 gamma2 k*).. *Max* (*encompassed gamma1 gamma2 k*)}

⟨*proof*⟩

lemma *encomp-at-idx-leq*:
  fixes *gamma1*::*int* ⇒ *real* **and** *gamma2*::*int* ⇒ *real*
  assumes *strict-mono* (*gamma1*::*int* ⇒ *real*)
  and *mono* (*gamma2*::*int* ⇒ *real*)
  and *encomp-at gamma1 gamma2 i k*
  and *gamma2 k′* ≤ *gamma1 i*
shows *k′* ≤ *k*
⟨*proof*⟩

lemma *encomp-at-unique*:
  assumes *strict-mono* (*gamma1*::*int* ⇒ *real*)
  and *mono* (*gamma2*::*int* ⇒ *real*)
  and *encomp-at gamma1 gamma2 i k*
  and *encomp-at gamma1 gamma2 i k′*
shows *k* = *k′*
⟨*proof*⟩

lemma *encomp-at-unique′*:
  assumes *strict-mono* (*gamma1*::*int* ⇒ *real*)
  and *mono* (*gamma2*::*int* ⇒ *real*)
  and *encomp-at gamma1 gamma2 i k*
  and *gamma2 k′* ≤ *gamma1 i*
  and *gamma1 i* < *gamma2* (*k′*+*1*)
shows *k* = *k′*
⟨*proof*⟩

lemma *encomp-at-refl*:
  fixes *gamma*::*′a*::{*one*, *plus*}⇒ *real*
  shows *encomp-at gamma gamma i i*
⟨*proof*⟩

3.2 Finer price grids

definition *finer-range*:: (*int* ⇒ *real*) ⇒ (*int* ⇒ *real*) ⇒ *bool* **where**
finer-range gamma1 gamma2 ≡ (∀ *i*. ∃ *k*. *encomp-at gamma1 gamma2 i k*)

definition *finer-grid* **where**
finer-grid P1 P2 ≡ *finer-range* (*grd P1*) (*grd P2*)

lemma *finer-grid-range*[*simp*]:
  assumes *finer-grid P1 P2*
  shows *finer-range* (*grd P1*) (*grd P2*)
  ⟨*proof*⟩

definition *coarse-idx* **where**
coarse-idx gamma1 gamma2 i =
  (*THE k*. *encomp-at gamma1 gamma2 i k*)

definition *finer-idx-bound* **where**
finer-idx-bound gamma1 gamma2 i =
(*THE k*. *gamma1 k* = *gamma2* (*coarse-idx gamma1 gamma2 i*))

lemma *finer-range-refl*:
shows *finer-range gamma gamma* ⟨*proof*⟩

locale *finer-ranges* =
fixes *gamma1*::*int* ⇒ *real* **and** *gamma2*::*int* ⇒ *real*
assumes *stm*: *strict-mono gamma1*
and *mon*: *mono gamma2*
and *fin*: *finer-range gamma1 gamma2*

begin

lemma *encomp-idx-unique*:
shows ∃!*k*. *encomp-at gamma1 gamma2 i k*
⟨*proof*⟩

lemma *coarse-idx-bounds*:
shows *encomp-at gamma1 gamma2 i* (*coarse-idx gamma1 gamma2 i*)
⟨*proof*⟩

lemma *encompassed-bounds*:
shows *i* ∈ *encompassed gamma1 gamma2* (*coarse-idx gamma1 gamma2 i*)
⟨*proof*⟩

lemma *encompassed-unique*:
assumes *i* ∈ *encompassed gamma1 gamma2 k*
shows *k* = *coarse-idx gamma1 gamma2 i*
⟨*proof*⟩

lemma *encompassed-inj*:
assumes *k*≠ *k′*
shows *encompassed gamma1 gamma2 k* ∩ *encompassed gamma1 gamma2 k′* =
{}
⟨*proof*⟩

lemma *coarse-idx-eq*:
assumes *gamma2 k′* ≤ *gamma1 i*
and *gamma1 i* < *gamma2* (*k′*+1)
shows *k′* = *coarse-idx gamma1 gamma2 i*
⟨*proof*⟩

lemma *coarse-idx-reached*:
assumes *gamma1 m* ≤ *gamma2 k*
and *gamma2 k* < *gamma1 M*
and *k* = *coarse-idx gamma1 gamma2 i*

shows $\exists j.\ gamma1\ j = gamma2\ k$
⟨*proof*⟩

lemma *coarse-idx-reached-unique*:
assumes $gamma1\ m \leq gamma2\ k$
and $gamma2\ k < gamma1\ M$
and $k = coarse\text{-}idx\ gamma1\ gamma2\ i$
shows $\exists! j.\ gamma1\ j = gamma2\ k$
⟨*proof*⟩

lemma *encomp-idx-mono*:
assumes $i < j$
and *encomp-at gamma1 gamma2 i k*
and *encomp-at gamma1 gamma2 j l*
and $k \neq l$
shows $k < l$
⟨*proof*⟩

lemma *encomp-idx-mono′*:
assumes $i \leq j$
and *encomp-at gamma1 gamma2 i k*
and *encomp-at gamma1 gamma2 j l*
shows $k \leq l$
⟨*proof*⟩

lemma *encomp-idx-mono-conv*:
assumes $k < l$
and *encomp-at gamma1 gamma2 i k*
and *encomp-at gamma1 gamma2 j l*
shows $i < j$
⟨*proof*⟩

lemma *finer-idx-bound-eq*:
assumes $gamma1\ m \leq gamma2\ (coarse\text{-}idx\ gamma1\ gamma2\ i)$
and $gamma2\ (coarse\text{-}idx\ gamma1\ gamma2\ i) < gamma1\ M$
shows $gamma1\ (finer\text{-}idx\text{-}bound\ gamma1\ gamma2\ i) =$
$gamma2\ (coarse\text{-}idx\ gamma1\ gamma2\ i)$
⟨*proof*⟩

lemma *finer-idx-bound-exists-eq*:
assumes $\exists m.\ gamma1\ m \leq gamma2\ (coarse\text{-}idx\ gamma1\ gamma2\ i)$
and $\exists M.\ gamma2\ (coarse\text{-}idx\ gamma1\ gamma2\ i) < gamma1\ M$
shows $gamma1\ (finer\text{-}idx\text{-}bound\ gamma1\ gamma2\ i) =$
$gamma2\ (coarse\text{-}idx\ gamma1\ gamma2\ i)$ ⟨*proof*⟩

lemma *finer-idx-bound-eq′*:
assumes $i \in encompassed\ gamma1\ gamma2\ k$
and $gamma1\ m \leq gamma2\ k$
and $gamma2\ k < gamma1\ M$

shows *gamma1* (*finer-idx-bound gamma1 gamma2 i*) = *gamma2 k*
⟨*proof*⟩

lemma *finer-idx-bound-exists-eq'*:
assumes *i* ∈ *encompassed gamma1 gamma2 k*
and ∃ *m*. *gamma1 m* ≤ *gamma2 k*
and ∃ *M*. *gamma2 k* < *gamma1 M*
shows *gamma1* (*finer-idx-bound gamma1 gamma2 i*) = *gamma2 k*
⟨*proof*⟩

lemma *finer-idx-bound-mem*:
assumes *gamma1 m* ≤ *gamma2* (*coarse-idx gamma1 gamma2 i*)
and *gamma2* (*coarse-idx gamma1 gamma2 i* + *1*) ≤ *gamma1 M*
and *gamma2* (*coarse-idx gamma1 gamma2 i*) ≠
gamma2 (*coarse-idx gamma1 gamma2 i* + *1*)
shows *finer-idx-bound gamma1 gamma2 i* ∈
encompassed gamma1 gamma2 (*coarse-idx gamma1 gamma2 i*)
⟨*proof*⟩

lemma *finer-idx-bound-reached*:
assumes *gamma1 m* ≤ *gamma2* (*coarse-idx gamma1 gamma2 i*)
and *gamma2* (*coarse-idx gamma1 gamma2 i*) < *gamma1 M*
and *gamma1 i* = *gamma2* (*coarse-idx gamma1 gamma2 i*)
shows *i* = *finer-idx-bound gamma1 gamma2 i*
⟨*proof*⟩

lemma *finer-idx-bound-leq*:
assumes *gamma1 m* ≤ *gamma2* (*coarse-idx gamma1 gamma2 i*)
and *gamma2* (*coarse-idx gamma1 gamma2 i*) < *gamma1 M*
shows *finer-idx-bound gamma1 gamma2 i* ≤ *i*
⟨*proof*⟩

lemma *finer-idx-bound-proj*:
assumes *i* ∈ *encompassed gamma1 gamma2 k*
and *j* ∈ *encompassed gamma1 gamma2 k*
and *gamma1 m* ≤ *gamma2 k*
and *gamma2 k* < *gamma1 M*
shows *finer-idx-bound gamma1 gamma2 i* = *finer-idx-bound gamma1 gamma2 j*
⟨*proof*⟩

lemma *finer-idx-bound-min*:
assumes *i* ∈ *encompassed gamma1 gamma2 k*
and *j* ∈ *encompassed gamma1 gamma2 k*
and *gamma1 m* ≤ *gamma2 k*
and *gamma2 k* < *gamma1 M*
shows *finer-idx-bound gamma1 gamma2 i* ≤ *j*
⟨*proof*⟩

lemma *coarse-idx-finer-bound*:

assumes *gamma1 m* $\leq$ *gamma2* (*coarse-idx gamma1 gamma2 i*)
and *gamma2* (*coarse-idx gamma1 gamma2 i*) $<$ *gamma1 M*
shows *coarse-idx gamma1 gamma2* (*finer-idx-bound gamma1 gamma2 i*) =
coarse-idx gamma1 gamma2 i
⟨*proof*⟩

lemma *finer-idx-bound-invol*:
assumes *gamma1 m* $\leq$ *gamma2* (*coarse-idx gamma1 gamma2 i*)
and *gamma2* (*coarse-idx gamma1 gamma2 i*) $<$ *gamma1 M*
shows *finer-idx-bound gamma1 gamma2* (*finer-idx-bound gamma1 gamma2 i*) =
finer-idx-bound gamma1 gamma2 i
⟨*proof*⟩

lemma *reached-imp-coarse*:
assumes *gamma1 i* = *gamma2 k*
and *gamma2 k* $\neq$ *gamma2* (*k*+*1*)
shows *gamma1* (*i*+*1*) $\leq$ *gamma2* (*k*+*1*)
⟨*proof*⟩

lemma *less-imp-coarse*:
assumes *gamma1 m* $<$ *gamma2 k*
and *gamma2 k* $\leq$ *gamma1 M*
and *gamma2 k* $\neq$ *gamma2* (*k*+*1*)
shows $\exists$ *i*. *encomp-at gamma1 gamma2 i k*
⟨*proof*⟩

lemma *ex-coarse-rep*:
assumes *gamma1 m* $\leq$ *gamma2 k*
and *gamma2 k* $\leq$ *gamma1 M*
and *gamma2 k* $\neq$ *gamma2* (*k*+*1*)
shows $\exists$ *i*. *encomp-at gamma1 gamma2 i k*
⟨*proof*⟩

lemma *encompassed-ne*:
assumes *gamma1 m* $\leq$ *gamma2 k*
and *gamma2 k* $\leq$ *gamma1 M*
and *gamma2 k* $\neq$ *gamma2* (*k*+*1*)
shows *encompassed gamma1 gamma2 k* $\neq$ {}
⟨*proof*⟩

lemma *encompassed-ne′*:
assumes $\exists$ *m*. *gamma1 m* $\leq$ *gamma2 k*
and $\exists$ *M*. *gamma2 k* $\leq$ *gamma1 M*
and *gamma2 k* $\neq$ *gamma2* (*k*+*1*)
shows *encompassed gamma1 gamma2 k* $\neq$ {}
⟨*proof*⟩

lemma *encompassed-finite*:
assumes *gamma1 m* $\leq$ *gamma2 k*

and *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$
shows *finite* (*encompassed gamma1 gamma2 k*)
⟨*proof*⟩

lemma *encompassed-finite'*:
assumes $\exists\, m.$ *gamma1* $m \leq$ *gamma2* k
and $\exists\, M.$ *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$
shows *finite* (*encompassed gamma1 gamma2 k*) ⟨*proof*⟩

lemma *encompassed-Min-in*:
assumes *gamma1* $m \leq$ *gamma2* k
and *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$
shows *Min* (*encompassed gamma1 gamma2 k*) $\in$ *encompassed gamma1 gamma2 k*
⟨*proof*⟩

lemma *encompassed-Max-in*:
assumes *gamma1* $m \leq$ *gamma2* k
and *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$
shows *Max* (*encompassed gamma1 gamma2 k*) $\in$ *encompassed gamma1 gamma2 k*
⟨*proof*⟩

lemma *encompassed-min-gamma-eq*:
assumes *gamma1* $m \leq$ *gamma2* k
and *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$
shows *gamma1* (*Min* (*encompassed gamma1 gamma2 k*)) = *gamma2* k
⟨*proof*⟩

lemma *encompassed-min-gamma-eq'*:
assumes $\exists\, m.$ *gamma1* $m \leq$ *gamma2* k
and $\exists\, M.$ *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$
shows *gamma1* (*Min* (*encompassed gamma1 gamma2 k*)) = *gamma2* k
⟨*proof*⟩

lemma *coarse-idx-upper*:
assumes *gamma2* $k <$ *gamma1* j
and $j \notin$ *encompassed gamma1 gamma2 k*
shows $k <$ *coarse-idx gamma1 gamma2 j*
⟨*proof*⟩

lemma *encompassed-max-Suc-eq*:
assumes *gamma1* $m \leq$ *gamma2* k
and *gamma2* $(k{+}1) \leq$ *gamma1* M
and *gamma2* $k \neq$ *gamma2* $(k{+}1)$

and *gamma2* $(k+1) \neq$ *gamma2* $(k+2)$
shows *Max* (*encompassed gamma1 gamma2 k*) + *1* $\in$
encompassed gamma1 gamma2 $(k+1)$
⟨*proof*⟩

lemma *encompassed-max-Suc-gamma-eq*:
assumes *gamma1 m* $\leq$ *gamma2 k*
and *gamma2* $(k+2) \leq$ *gamma1 M*
and *gamma2 k* $\neq$ *gamma2* $(k+1)$
and *gamma2* $(k+1) \neq$ *gamma2* $(k+2)$
shows *gamma1* (*Max* (*encompassed gamma1 gamma2 k*) + *1*) = *gamma2* $(k+1)$
⟨*proof*⟩

lemma *encompassed-max-Suc-gamma-eq′*:
assumes $\exists$ *m*. *gamma1 m* $\leq$ *gamma2 k*
and $\exists$ *M*. *gamma2* $(k+2) \leq$ *gamma1 M*
and *gamma2 k* $\neq$ *gamma2* $(k+1)$
and *gamma2* $(k+1) \neq$ *gamma2* $(k+2)$
shows *gamma1* (*Max* (*encompassed gamma1 gamma2 k*) + *1*) = *gamma2* $(k+1)$
⟨*proof*⟩

end

lemma *coarse-idx-refl*:
fixes *gamma*::*int* $\Rightarrow$ *real*
assumes *strict-mono gamma*
shows *i* = *coarse-idx gamma gamma i*
⟨*proof*⟩

3.3 Pools with finer grids and coinciding profiles

definition *pool-coarse-idx* **where**
pool-coarse-idx = (λ*P1 P2 i*. *coarse-idx* (*grd P1*) (*grd P2*) *i*)

lemma *pool-coarse-idxD*:
assumes *k* = *pool-coarse-idx P1 P2 i*
shows *k* = *coarse-idx* (*grd P1*) (*grd P2*) *i*
⟨*proof*⟩

definition *pool-finer-idx-bound* **where**
pool-finer-idx-bound = (λ*P1 P2 i*. *finer-idx-bound* (*grd P1*) (*grd P2*) *i*)

lemma *pool-finer-idx-boundD*:
assumes *l* = *pool-finer-idx-bound P1 P2 i*
shows *l* = *finer-idx-bound* (*grd P1*) (*grd P2*) *i*
⟨*proof*⟩

definition *finer-pool* **where**
finer-pool P1 P2 $\equiv$ *finer-grid P1 P2* $\wedge$

$(\forall\, i.\ lq\ P1\ i = lq\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i)) \wedge$
$(\forall\, i.\ fee\ P1\ i = fee\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i))$

lemma *finer-poolI*[*intro*]:
assumes *finer-range* (*grd P1*) (*grd P2*)
and $(\forall\, i.\ lq\ P1\ i = lq\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i))$
and $(\forall\, i.\ fee\ P1\ i = fee\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i))$
shows *finer-pool P1 P2*
⟨*proof*⟩

lemma *finer-poolD*:
assumes *finer-pool P1 P2* **shows**
finer-range (*grd P1*) (*grd P2*)
$\forall\, i.\ lq\ P1\ i = lq\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i)$
$\forall\, i.\ fee\ P1\ i = fee\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i)$
⟨*proof*⟩

lemma *finer-pool-refl*:
assumes *strict-mono* (*grd P*)
shows *finer-pool P P*
⟨*proof*⟩

locale *finer-pools* =
fixes *P1 P2*
assumes *fin-pool*: *finer-pool P1 P2*

begin

lemma *finer-pool-grid*:
shows *finer-range* (*grd P1*) (*grd P2*) ⟨*proof*⟩

lemma *finer-pool-liq*:
shows $\forall\, i.\ lq\ P1\ i = lq\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i)$
⟨*proof*⟩

lemma *finer-pool-fee*:
shows $\forall\, i.\ fee\ P1\ i = fee\ P2\ (pool\text{-}coarse\text{-}idx\ P1\ P2\ i)$
⟨*proof*⟩

lemma *encompassed-liq-eq*:
assumes *strict-mono* (*grd P1*)
and *mono* (*grd P2*)
and $i \in encompassed\ (grd\ P1)\ (grd\ P2)\ k$
shows $lq\ P1\ i = lq\ P2\ k$
⟨*proof*⟩

lemma *encompassed-fee-eq*:
assumes *strict-mono* (*grd P1*)
and *mono* (*grd P2*)

and $i \in$ *encompassed* (*grd P1*) (*grd P2*) k
shows *fee P1 i* = *fee P2 k*
⟨*proof*⟩

lemma *sum-rng-token*:
assumes *strict-mono* (*grd P1*)
and *mono* (*grd P2*)
and *grd P1 m1* ≤ *grd P2 k*
and *grd P2* (*k*+*1*) ≤ *grd P1 M1*
and *grd P2 k* ≠ *grd P2* (*k* + *1*)
and ⋀ *a b*. *a* ∈ *encompassed* (*grd P1*) (*grd P2*) *b* ⟹
g (*lq P1*) *a* = *g*′ (*lq P2*) *b*
and ∀ *i* ∈ *encompassed* (*grd P1*) (*grd P2*) *k*. *dff x i* = *f* (*i*+*1*) − *f i*
shows *sum* (*rng-token dff* (*g* (*lq P1*)) *x*)
(*encompassed* (*grd P1*) (*grd P2*) *k*) =
(*g*′ (*lq P2*)) *k* ∗ (*f* (*Max* (*encompassed* (*grd P1*) (*grd P2*) *k*) + *1*) −
f (*Min* (*encompassed* (*grd P1*) (*grd P2*) *k*)))
⟨*proof*⟩

lemma *sum-rng-gen-quote*:
assumes *strict-mono* (*grd P1*)
and *mono* (*grd P2*)
and *grd P1 m1* ≤ *grd P2 k*
and *grd P2* (*k*+*2*) ≤ *grd P1 M1*
and *grd P2 k* ≠ *grd P2* (*k* + *1*)
and *grd P2* (*k*+*1*) ≠ *grd P2* (*k* + *2*)
and ⋀ *a b*. *a* ∈ *encompassed* (*grd P1*) (*grd P2*) *b* ⟹
f (*lq P1*) *a* = *f*′ (*lq P2*) *b*
shows *sum* (*rng-gen-quote* (*grd P1*) (*f* (*lq P1*)) *x*)
(*encompassed* (*grd P1*) (*grd P2*) *k*) =
rng-gen-quote (*grd P2*) (*f*′ (*lq P2*)) *x k*
⟨*proof*⟩

lemma *sum-rng-gen-base*:
assumes *strict-mono* (*grd P1*)
and *mono* (*grd P2*)
and *grd P1 m1* ≤ *grd P2 k*
and *grd P2* (*k*+*2*) ≤ *grd P1 M1*
and *grd P2 k* ≠ *grd P2* (*k* + *1*)
and *grd P2* (*k*+*1*) ≠ *grd P2* (*k* + *2*)
and ⋀ *a b*. *a* ∈ *encompassed* (*grd P1*) (*grd P2*) *b* ⟹
f (*lq P1*) *a* = *f*′ (*lq P2*) *b*
shows *sum* (*rng-gen-base* (*grd P1*) (*f* (*lq P1*)) *x*)
(*encompassed* (*grd P1*) (*grd P2*) *k*) =
rng-gen-base (*grd P2*) (*f*′ (*lq P2*)) *x k*
⟨*proof*⟩

lemma *finer-imp-finite-liq*:
assumes *strict-mono* (*grd P1*)

and *mono* (*grd P2*)
and *finite-liq P2*
and $\bigwedge k.$ *lq P2* $k \neq 0 \Longrightarrow$ *finite* (*encompassed* (*grd P1*) (*grd P2*) *k*)
shows *finite-liq P1*
⟨*proof*⟩

lemma *finer-imp-finite-liq′*:
assumes *finer-pool P1 P2*
and *strict-mono* (*grd P1*)
and *mono* (*grd P2*)
and *finite-liq P1*
and *finite* {*k*. *encompassed* (*grd P1*) (*grd P2*) *k* = {}}
shows *finite-liq P2*
⟨*proof*⟩

end

3.4 Spanning grids

definition *span-grid* **where**
span-grid P ⟷ *strict-mono* (*grd P*) ∧ (∀ *i*. *0* < *grd P i*) ∧
(∀ *r*>*0*. ∃ *i*. *grd P i* < *r*) ∧ (∀ *r*. ∃ *i*. *r* < *grd P i*)

lemma *span-gridD*:
assumes *span-grid P*
shows *strict-mono* (*grd P*) ∀ *i*. *0* < *grd P i*
∀ *r*>*0*. ∃ *i*. *grd P i* < *r* ∀ *r*. ∃ *i*. *r* < *grd P i*
⟨*proof*⟩

lemma *span-gridI*[*intro*]:
assumes *strict-mono* (*grd P*)
and ∀ *i*. *0* < *grd P i*
and ∀ *r*>*0*. ∃ *i*. *grd P i* < *r*
and ∀ *r*. ∃ *i*. *r* < *grd P i*
shows *span-grid P* ⟨*proof*⟩

lemma *span-grid-eq*:
assumes *span-grid P*
and *grd P* = *grd P′*
shows *span-grid P′* ⟨*proof*⟩

locale *finer-spanning-pool* = *finer-pools* +
assumes *span*: *span-grid P1*

begin

lemma *finer-spanning-gt*:
shows ∃ *i*. *r* < *grd P2 i*
⟨*proof*⟩

lemma *finer-spanning-lt*:
assumes $0 < r$
shows $\exists\, i.\ grd\ P2\ i < r$
⟨*proof*⟩

lemma *finer-span-grid*:
assumes $\forall\, i.\ 0 < grd\ P2\ i$
and *strict-mono* $(grd\ P2)$
shows *span-grid P2*
⟨*proof*⟩

end

locale *finer-two-spanning-pools* = *finer-spanning-pool* +
assumes *span2*: *span-grid P2*

sublocale *finer-two-spanning-pools* ⊆ *finer-ranges grd P1 grd P2*
⟨*proof*⟩

context *finer-two-spanning-pools*
begin

lemma *spanning-sum-rng-gen-quote*:
assumes $\bigwedge\ a\ b.\ \ a \in encompassed\ (grd\ P1)\ (grd\ P2)\ b \Longrightarrow$
$f\ (lq\ P1)\ a = f'\ (lq\ P2)\ b$
shows *sum* $(rng\text{-}gen\text{-}quote\ (grd\ P1)\ (f\ (lq\ P1))\ x)$
$(encompassed\ (grd\ P1)\ (grd\ P2)\ k) =$
$rng\text{-}gen\text{-}quote\ (grd\ P2)\ (f'\ (lq\ P2))\ x\ k$
⟨*proof*⟩

lemma *spanning-sum-rng-gen-base*:
assumes $\bigwedge\ a\ b.\ \ a \in encompassed\ (grd\ P1)\ (grd\ P2)\ b \Longrightarrow$
$f\ (lq\ P1)\ a = f'\ (lq\ P2)\ b$
shows *sum* $(rng\text{-}gen\text{-}base\ (grd\ P1)\ (f\ (lq\ P1))\ x)$
$(encompassed\ (grd\ P1)\ (grd\ P2)\ k) =$
$rng\text{-}gen\text{-}base\ (grd\ P2)\ (f'\ (lq\ P2))\ x\ k$
⟨*proof*⟩

lemma *span-grid-encompassed*:
shows *finite* $(encompassed\ (grd\ P1)\ (grd\ P2)\ k)$
⟨*proof*⟩

lemma *span-grids-finite-liq*:
assumes *finite-liq P2*
shows *finite-liq P1*
⟨*proof*⟩

lemma *span-grids-ex-le*:

shows $\exists\, m.\ grd\ P1\ m \leq grd\ P2\ k$
⟨*proof*⟩

lemma *span-grids-ex-ge*:
shows $\exists\, M.\ grd\ P2\ k \leq grd\ P1\ M$
⟨*proof*⟩

lemma *span-grids-encompassed-ne*:
shows *encompassed* (*grd P1*) (*grd P2*) $k \neq \{\}$
⟨*proof*⟩

end

3.5 Spanning grids and finite liquidity

locale *finer-two-span-finite-liq* = *finer-two-spanning-pools* +
assumes *fin-liq*: *finite-liq P1*

sublocale *finer-two-span-finite-liq* ⊆ *finite-liq-pool P1*
⟨*proof*⟩

lemma (**in** *finer-two-span-finite-liq*) *span-grids-finite-liq'*:
shows *finite-liq P2*
⟨*proof*⟩

sublocale *finer-two-span-finite-liq* ⊆ *finite-liq-pool P2*
⟨*proof*⟩

context *finer-two-span-finite-liq*
begin

lemma *finer-pool-encompassed-Union*:
shows $(\bigcup\ (encompassed\ (grd\ P1)\ (grd\ P2)\ `\{i.\ lq\ P2\ i \neq 0\})) =$
$\{i.\ lq\ P1\ i \neq 0\}$
⟨*proof*⟩

lemma *spanning-finer-gen-quote-eq*:
assumes $\bigwedge\ a\ b.\ \ a \in encompassed\ (grd\ P1)\ (grd\ P2)\ b \Longrightarrow$
$f\ (lq\ P1)\ a = f'\ (lq\ P2)\ b$
and $\bigwedge i.\ lq\ P2\ i = 0 \Longrightarrow f'\ (lq\ P2)\ i = 0$
and $\bigwedge i.\ lq\ P1\ i = 0 \Longrightarrow f\ (lq\ P1)\ i = 0$
shows *gen-quote* (*grd P1*) (f (*lq P1*)) x = *gen-quote* (*grd P2*) (f' (*lq P2*)) x
⟨*proof*⟩

lemma *spanning-finer-gen-base-eq*:
assumes $\bigwedge\ a\ b.\ \ a \in encompassed\ (grd\ P1)\ (grd\ P2)\ b \Longrightarrow$
$f\ (lq\ P1)\ a = f'\ (lq\ P2)\ b$
and $\bigwedge i.\ lq\ P2\ i = 0 \Longrightarrow f'\ (lq\ P2)\ i = 0$
and $\bigwedge i.\ lq\ P1\ i = 0 \Longrightarrow f\ (lq\ P1)\ i = 0$

shows *gen-base* (*grd P1*) (*f* (*lq P1*)) *x* = *gen-base* (*grd P2*) (*f'* (*lq P2*)) *x*
⟨*proof*⟩

end

end
theory *CLMM-Description* **imports** *Grid-Information*

begin

4 CLMM description

Definition of CLMMs (Concentrated Liquidity Market Makers)

4.1 Preliminary results

definition *clmm-dsc* **where**
clmm-dsc P ⟷ (*span-grid P*) ∧ (*finite-liq P*) ∧ (∀ *i*. *0* ≤ *lq P i*) ∧
(∀ *i*. *0* ≤ *fee P i*) ∧ (∀ *i*. *fee P i* < *1*)

lemma *clmm-dscI*[*intro*]:
assumes *span-grid P*
and *finite-liq P*
and ∀ *i*. *0* ≤ *lq P i*
and ∀ *i*. *0* ≤ *fee P i*
and ∀ *i*. *fee P i* < *1*
shows *clmm-dsc P* ⟨*proof*⟩

lemma *clmm-dsc-span-grid*:
assumes *clmm-dsc P*
shows *span-grid P* ⟨*proof*⟩

lemma *clmm-dsc-grid*[*simp*]:
assumes *clmm-dsc P*
shows *strict-mono* (*grd P*) (∀ *i*. *0* < *grd P i*)
(∀ *r*>*0*. ∃ *i*. *grd P i* < *r*) (∀ *r*. ∃ *i*. *r* < *grd P i*)
⟨*proof*⟩

lemma *clmm-dsc-grd-Suc*:
assumes *clmm-dsc P*
shows *grd P i* < *grd P* (*i*+*1*) ⟨*proof*⟩

lemma *clmm-dsc-grd-smono*:
assumes *clmm-dsc P*

and $i < j$
shows *grd P i < grd P j* ⟨*proof*⟩

lemma *clmm-dsc-grd-mono*:
assumes *clmm-dsc P*
and $i \leq j$
shows *grd P i* ≤ *grd P j* ⟨*proof*⟩

lemma *clmm-dsc-liq*:
assumes *clmm-dsc P*
shows *finite-liq P 0* ≤ *lq P i* ⟨*proof*⟩

lemma *clmm-dsc-fees*:
assumes *clmm-dsc P*
shows (∀ *i*. *0* ≤ *fee P i*) ∧ (∀ *i*. *fee P i < 1*) ⟨*proof*⟩

lemma *clmm-dsc-fees-neq-1*:
assumes *clmm-dsc P*
shows ∀ *i*. *fee P i* ≠ *1*
⟨*proof*⟩

lemma *clmm-dsc-gross-liq*:
assumes *clmm-dsc P*
shows *nz-support* (*gross-fct* (*lq P*) (*fee P*)) = *nz-support* (*lq P*)
⟨*proof*⟩

lemma *clmm-dsc-gross-liq-zero-iff*:
assumes *clmm-dsc P*
shows (*lq P i = 0*) ⟷ (*gross-fct* (*lq P*) (*fee P*) *i = 0*)
⟨*proof*⟩

lemma *gross-liq-gt*:
assumes *clmm-dsc P*
and *lq P i* ≠ *0*
and *L = gross-fct* (*lq P*) (*fee P*)
shows *0 < L i* ⟨*proof*⟩

lemma *gross-liq-ge*:
assumes *clmm-dsc P*
and *L = gross-fct* (*lq P*) (*fee P*)
shows *0* ≤ *L i* ⟨*proof*⟩

lemma *rng-quote-net-ge*:
assumes *clmm-dsc P*
shows *0* ≤ *lq P i* ∗ (*grd P* (*i*+*1*) − *grd P i*)
⟨*proof*⟩

lemma *rng-quote-gross-ge*:
assumes *clmm-dsc P*

and $L = gross\text{-}fct\ (lq\ P)\ (fee\ P)$
shows $0 \leq L\ i * (grd\ P\ (i+1) - grd\ P\ i)$
⟨*proof*⟩

lemma *clmm-quote-gross-pos*:
assumes *clmm-dsc P*
shows $0 \leq quote\text{-}gross\ P\ sqp$ ⟨*proof*⟩

lemma *clmm-quote-gross-mono*:
assumes *clmm-dsc P*
shows *mono* (*quote-gross P*)
⟨*proof*⟩

lemma *quote-gross-imp-sqp-lt*:
assumes *clmm-dsc P*
and $quote\text{-}gross\ P\ sqp < quote\text{-}gross\ P\ sqp'$
shows $sqp < sqp'$
⟨*proof*⟩

lemma *clmm-quote-net-mono*:
assumes *clmm-dsc P*
shows *mono* (*quote-net P*)
⟨*proof*⟩

lemma *clmm-base-gross-antimono*:
assumes *clmm-dsc P*
shows *antimono* (*base-gross P*)
⟨*proof*⟩

lemma *clmm-base-net-antimono*:
assumes *clmm-dsc P*
shows *antimono* (*base-net P*)
⟨*proof*⟩

lemma *liq-grd-min*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
shows $0 < grd\text{-}min\ P$ ⟨*proof*⟩

lemma *liq-grd-min-max*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
shows $grd\text{-}min\ P < grd\text{-}max\ P$
⟨*proof*⟩

definition *rng-blw* **where**
$rng\text{-}blw\ P\ prc = \{i.\ grd\ P\ i \leq prc\}$

lemma *rng-blw-mem*[*simp*]:

assumes $i \in$ *rng-blw P prc*
shows *grd P i* $\leq$ *prc* ⟨*proof*⟩

lemma *rng-blw-bdd-above*:
assumes *clmm-dsc P*
shows *bdd-above* (*rng-blw P prc*) ⟨*proof*⟩

lemma *rng-blw-ne*:
assumes *clmm-dsc P*
and *0* $<$ *prc*
shows *rng-blw P prc* $\neq$ {}
⟨*proof*⟩

definition *lower-tick* **where**
lower-tick P prc = *Sup* (*rng-blw P prc*)

lemma *grd-lower-tick-cong*:
assumes *grd P1* = *grd P2*
shows *lower-tick P1 sqp* = *lower-tick P2 sqp*
⟨*proof*⟩

lemma *lower-tick-mem*:
assumes *clmm-dsc P*
and *0* $<$ *prc*
shows *lower-tick P prc* $\in$ *rng-blw P prc* ⟨*proof*⟩

lemma *lower-tick-geq*:
assumes *clmm-dsc P*
and *0* $<$ *prc*
shows *grd P* (*lower-tick P prc*) $\leq$ *prc*
⟨*proof*⟩

lemma *lower-tick-geq′*:
assumes *clmm-dsc P*
and $i \in$ *rng-blw P prc*
shows $i \leq$ *lower-tick P prc* ⟨*proof*⟩

lemma *lower-tick-ubound*:
assumes *clmm-dsc P*
and i = *lower-tick P prc*
shows *prc* $<$ *grd P* (*i*+*1*)
⟨*proof*⟩

lemma *lower-tick-lbound*:
assumes *clmm-dsc P*
and *0* $<$ *prc*
and i = *lower-tick P prc*
shows *grd P i* $\leq$ *prc* ⟨*proof*⟩

lemma *lower-tick-lt*:
  assumes *clmm-dsc P*
  and *0 < sqp'*
  and *i = lower-tick P sqp*
  and *j = lower-tick P sqp'*
  and *i < j*
shows *sqp < sqp'*
⟨*proof*⟩

lemma *lower-tick-lt'*:
  assumes *clmm-dsc P*
  and *0 < sqp'*
  and *i = lower-tick P sqp*
  and *j = lower-tick P sqp'*
  and *sqp' < sqp*
  and *grd P i = sqp*
shows *j < i*
⟨*proof*⟩

lemma *lower-tick-mono*:
  assumes *clmm-dsc P*
  and *0 < sqp*
  and *i = lower-tick P sqp*
  and *j = lower-tick P sqp'*
  and *sqp ≤ sqp'*
shows *i ≤ j*
  ⟨*proof*⟩

lemma *lower-tick-eq*:
  assumes *clmm-dsc P*
  and *grd P i = sqp*
shows *lower-tick P sqp = i*
⟨*proof*⟩

lemma *lower-tick-charact*:
  assumes *clmm-dsc P*
  and *grd P i ≤ sqp*
  and *sqp < grd P (i+1)*
shows *lower-tick P sqp = i*
⟨*proof*⟩

lemma *lower-tick-grd-min*:
  assumes *strict-mono (grd P)*
shows *idx-min (lq P) = lower-tick P (grd-min P)*
⟨*proof*⟩

lemma *lower-tick-grd-max*:
  assumes *strict-mono (grd P)*
  shows *idx-max (lq P) + 1 = lower-tick P (grd-max P)*

⟨*proof*⟩

lemma *grd-max-gt-if*:
  assumes *clmm-dsc P*
  and *i* = *lower-tick P sqp*
  and *lq P i* ≠ *0*
shows *sqp* < *grd-max P*
⟨*proof*⟩

4.2 Quote token addition and withdrawal in a CLMM

lemma (**in** *finite-nz-support*) *clmm-gen-quote-sum*:
  assumes *clmm-dsc P*
  and *0* < *sqp*
  and *j* = *lower-tick P sqp*
shows *gen-quote* (*grd P*) *L sqp* =
    L j ∗ (*sqp* − *grd P j*) +
    sum (λ *i*. *L i* ∗ (*grd P* (*i*+*1*) − *grd P i*)) {*i*. *L i* ≠ *0* ∧ *i* < *j*}
⟨*proof*⟩

lemma *clmm-gen-quote-grd-min*:
  assumes *clmm-dsc P*
  and *nz-support L* ≠ {}
  and *finite* (*nz-support L*)
  and *nz-support L* = *nz-support* (*lq P*)
shows *gen-quote* (*grd P*) *L* (*grd-min P*) = *0* ⟨*proof*⟩

lemma (**in** *finite-nz-support*) *clmm-gen-quote-grd-min-le*:
  assumes *clmm-dsc P*
  and *nz-support L* = *nz-support* (*lq P*)
  and *sqp* ≤ *grd-min P*
  and *0* < *sqp*
shows *gen-quote* (*grd P*) *L sqp* = *0*
⟨*proof*⟩

lemma *clmm-quote-gross-sum*:
  assumes *clmm-dsc P*
  and *0* < *sqp*
  and *L* = *gross-fct* (*lq P*) (*fee P*)
  and *j* = *lower-tick P sqp*
shows *quote-gross P sqp* =
    L j ∗ (*sqp* − *grd P j*) +
    sum (λ *i*. *L i* ∗ (*grd P* (*i*+*1*) − *grd P i*)) {*i*. *L i* ≠ *0* ∧ *i* < *j*}
⟨*proof*⟩

lemma *clmm-quote-gross-grd-min*:
  assumes *clmm-dsc P*
  and *nz-support* (*lq P*) ≠ {}
shows *quote-gross P* (*grd-min P*) = *0* ⟨*proof*⟩

lemma *clmm-quote-gross-grd-min-le*:
assumes *clmm-dsc P*
and *sqp* $\leq$ *grd-min P*
and $0 < sqp$
shows *quote-gross P sqp* $= 0$ ⟨*proof*⟩

lemma *clmm-quote-reach-zero*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
shows *quote-reach P 0* $=$ *grd-min P*
⟨*proof*⟩

lemma *clmm-quote-reach-ge*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
and $0 \leq y$
and $y \leq$ *quote-gross P* (*grd-max P*)
shows *grd-min P* $\leq$ (*quote-reach P y*)
⟨*proof*⟩

lemma *clmm-quote-reach-pos*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
and $0 \leq y$
and $y \leq$ *quote-gross P* (*grd-max P*)
and *sqp* $=$ *quote-reach P y*
shows $0 < sqp$
⟨*proof*⟩

lemma *clmm-quote-reach-mem*:
assumes *clmm-dsc P*
and $0 \leq y$
and $y \leq$ *quote-gross P* (*grd-max P*)
and *nz-support* (*lq P*) $\neq \{\}$
shows *quote-reach P y* $\in$ *quote-gross P* $-$ ‘ $\{y\}$
⟨*proof*⟩

lemma *clmm-quote-reach-le*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
and $0 < y$
and *sqp* $\in$ *quote-gross P* $-$‘$\{y\}$
and $sqp' =$ *quote-reach P y*
shows $sqp' \leq sqp$
⟨*proof*⟩

lemma *clmm-quote-net-sum*:
assumes *clmm-dsc P*

and $0 < sqp$
and $L = lq\ P$
and $j = lower\text{-}tick\ P\ sqp$
shows $quote\text{-}net\ P\ sqp =$
$L\ j * (sqp - grd\ P\ j) +$
$sum\ (\lambda\ i.\ L\ i * (grd\ P\ (i{+}1) - grd\ P\ i))\ \{i.\ L\ i \neq 0 \wedge i < j\}$
⟨*proof*⟩

lemma *clmm-quote-net-grd-min*:
assumes *clmm-dsc P*
and *nz-support* $(lq\ P) \neq \{\}$
shows *quote-net P* $(grd\text{-}min\ P) = 0$ ⟨*proof*⟩

lemma *clmm-quote-gross-reach-eq*:
assumes *clmm-dsc P*
and *nz-support* $(lq\ P) \neq \{\}$
and $0 \leq y$
and $y \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
shows *quote-gross P* $(quote\text{-}reach\ P\ y) = y$
⟨*proof*⟩

definition *gen-quote-diff* **where**
$gen\text{-}quote\text{-}diff\ P\ L\ sqp\ sqp' = gen\text{-}quote\ (grd\ P)\ L\ sqp' - gen\text{-}quote\ (grd\ P)\ L\ sqp$

lemma (**in** *finite-nz-support*) *clmm-gen-quote-diff-eq*:
assumes *clmm-dsc P*
and $j = lower\text{-}tick\ P\ sqp$
and $k = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
shows $gen\text{-}quote\text{-}diff\ P\ L\ sqp\ sqp' = L\ k * (sqp' - grd\ P\ k) +$
$sum\ (\lambda\ i.\ L\ i * (grd\ P\ (i{+}1) - grd\ P\ i))\ \{i.\ L\ i \neq 0 \wedge j < i \wedge i < k\} +$
$L\ j * (grd\ P\ (j{+}1) - sqp)$
⟨*proof*⟩

lemma (**in** *finite-nz-support*) *clmm-gen-quote-diff-eq'*:
assumes *clmm-dsc P*
and $j = lower\text{-}tick\ P\ sqp$
and $j = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
and $L'\ j = L\ j$
shows $gen\text{-}quote\text{-}diff\ P\ L\ sqp\ sqp' = L'\ j * (sqp' - sqp)$
⟨*proof*⟩

lemma *clmm-quote-gross-diff-eq*:
assumes *clmm-dsc P*
and $L = gross\text{-}fct\ (lq\ P)\ (fee\ P)$

and $j = \textit{lower-tick}\ P\ sqp$
and $k = \textit{lower-tick}\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
shows $\textit{quote-gross}\ P\ sqp' - \textit{quote-gross}\ P\ sqp = L\ k * (sqp' - \textit{grd}\ P\ k) +$
$\textit{sum}\ (\lambda\ i.\ L\ i * (\textit{grd}\ P\ (i+1) - \textit{grd}\ P\ i))\ \{i.\ L\ i \neq 0 \wedge j < i \wedge i < k\} +$
$L\ j * (\textit{grd}\ P\ (j+1) - sqp)$
⟨*proof*⟩

lemma *clmm-rng-quote-strict-pos*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $L\ i \neq 0$
shows $0 < L\ i * (\textit{grd}\ P\ (i+1) - \textit{grd}\ P\ i)$ ⟨*proof*⟩

lemma *clmm-sum-rng-quote-pos*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
shows $0 \leq \textit{sum}\ (\lambda\ i.\ L\ i * (\textit{grd}\ P\ (i+1) - \textit{grd}\ P\ i))\ M$
⟨*proof*⟩

lemma *clmm-sum-rng-quote-strict-pos*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $L\ i \neq 0$
and $i \in M$
and *finite M*
shows $0 < \textit{sum}\ (\lambda\ i.\ L\ i * (\textit{grd}\ P\ (i+1) - \textit{grd}\ P\ i))\ M$
⟨*proof*⟩

lemma *clmm-quote-gross-eq-sum-only-if*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $j = \textit{lower-tick}\ P\ sqp$
and $k = \textit{lower-tick}\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
and $\textit{quote-gross}\ P\ sqp' = \textit{quote-gross}\ P\ sqp$
and $j < i$
and $i < k$
shows $L\ i = 0$
⟨*proof*⟩

lemma *clmm-quote-gross-eq-sum-emp*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $j = \textit{lower-tick}\ P\ sqp$

and $k = \textit{lower-tick}\ P\ \textit{sqp}'$
and $0 < \textit{sqp}$
and $\textit{sqp} \leq \textit{sqp}'$
and $j < k$
and $\textit{quote-gross}\ P\ \textit{sqp}' = \textit{quote-gross}\ P\ \textit{sqp}$
shows $\{i.\ L\ i \neq 0 \wedge j < i \wedge i < k\} = \{\}$
⟨*proof*⟩

lemma *clmm-quote-gross-eq-lower-only-if*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $j = \textit{lower-tick}\ P\ \textit{sqp}$
and $k = \textit{lower-tick}\ P\ \textit{sqp}'$
and $0 < \textit{sqp}$
and $\textit{sqp} \leq \textit{sqp}'$
and $j < k$
and $\textit{quote-gross}\ P\ \textit{sqp}' = \textit{quote-gross}\ P\ \textit{sqp}$
shows $L\ j = 0$
⟨*proof*⟩

lemma *clmm-quote-gross-eq-upper-only-if*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $j = \textit{lower-tick}\ P\ \textit{sqp}$
and $k = \textit{lower-tick}\ P\ \textit{sqp}'$
and $0 < \textit{sqp}$
and $\textit{sqp} \leq \textit{sqp}'$
and $j < k$
and $\textit{quote-gross}\ P\ \textit{sqp}' = \textit{quote-gross}\ P\ \textit{sqp}$
shows $L\ k = 0 \vee \textit{grd}\ P\ k = \textit{sqp}'$
⟨*proof*⟩

lemma *clmm-quote-gross-diff-eq'*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $j = \textit{lower-tick}\ P\ \textit{sqp}$
and $j = \textit{lower-tick}\ P\ \textit{sqp}'$
and $0 < \textit{sqp}$
and $\textit{sqp} \leq \textit{sqp}'$
shows $\textit{quote-gross}\ P\ \textit{sqp}' - \textit{quote-gross}\ P\ \textit{sqp} = L\ j * (\textit{sqp}' - \textit{sqp})$
⟨*proof*⟩

lemma *clmm-quote-gross-eq-lower-only-if'*:
assumes *clmm-dsc P*
and $L = \textit{gross-fct}\ (\textit{lq}\ P)\ (\textit{fee}\ P)$
and $j = \textit{lower-tick}\ P\ \textit{sqp}$
and $j = \textit{lower-tick}\ P\ \textit{sqp}'$
and $0 < \textit{sqp}$
and $\textit{sqp} < \textit{sqp}'$

and *quote-gross P sqp′* = *quote-gross P sqp*
shows *L j* = *0*
⟨*proof*⟩

lemma *clmm-quote-reach-grd-liq*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *0* < *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
and *j* = *lower-tick P sqp*
and *grd P j* = *sqp*
and *sqp* = *quote-reach P y*
shows *lq P* (*j* − *1*) ≠ *0*
⟨*proof*⟩

lemma *quote-gross-gt-grd-min*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *grd-min P* < *sqp*
shows *0* < *quote-gross P sqp*
⟨*proof*⟩

lemma *quote-gross-pos-gt-grd-min*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *0* < *quote-gross P sqp*
shows *grd-min P* < *sqp*
⟨*proof*⟩

lemma *quote-gross-disj-gt*:
assumes *clmm-dsc P*
and *i* = *lower-tick P sqp*
and *j* = *lower-tick P sqp′*
and *i* ≤ *k*
and *k* < *j*
and *lq P k* ≠ *0*
and *0* < *sqp*
and *0* < *sqp′*
shows *quote-gross P sqp* < *quote-gross P sqp′*
⟨*proof*⟩

lemma *quote-gross-disj-gt′*:
assumes *clmm-dsc P*
and *i* = *lower-tick P sqp*
and *j* = *lower-tick P sqp′*
and *i* < *j*
and *lq P j* ≠ *0*
and *grd P j* < *sqp′*
and *0* < *sqp*

and $0 < sqp'$
shows *quote-gross P sqp* < *quote-gross P sqp'*
⟨*proof*⟩

lemma *quote-gross-lower-eq-gt*:
assumes *clmm-dsc P*
and $j =$ *lower-tick P sqp*
and $j =$ *lower-tick P sqp'*
and *lq P j* $\neq 0$
and $0 <$ *sqp*
and *sqp* < *sqp'*
shows *quote-gross P sqp* < *quote-gross P sqp'*
⟨*proof*⟩

lemma *quote-reach-gt-grd-min*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
and $0 < y$
and $y \leq$ *quote-gross P* (*grd-max P*)
and *sqp* = *quote-reach P y*
shows *grd-min P* < *sqp*
⟨*proof*⟩

lemma *sqp-lt-grd-max-imp-idx*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
and $0 <$ *sqp*
and *sqp* < *grd-max P*
and $i =$ *lower-tick P sqp*
shows $i \leq$ *idx-max* (*lq P*)
⟨*proof*⟩

lemma *quote-gross-lt-grd-max*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq \{\}$
and $0 <$ *sqp*
and *sqp* < *grd-max P*
shows *quote-gross P sqp* < *quote-gross P* (*grd-max P*)
⟨*proof*⟩

lemma *idx-max-gt-liq*:
assumes *clmm-dsc P*
and $j =$ *idx-max* (*lq P*)
shows $\forall\, k > j.$ *lq P k* $= 0$
⟨*proof*⟩

lemma *idx-min-lt-liq*:
assumes *clmm-dsc P*
and $j =$ *idx-min* (*lq P*)

shows $\forall\, k < j.\ lq\ P\ k = 0$
⟨*proof*⟩

lemma *quote-reach-le'*:
  assumes *clmm-dsc P*
  and *grd-min P < sqp*
  and *i = lower-tick P sqp*
  and *lq P i ≠ 0*
  and *y = quote-gross P sqp*
shows *quote-reach P y ≤ sqp*
⟨*proof*⟩

lemma *quote-reach-gross-le*:
  assumes *clmm-dsc P*
  and *grd-min P ≤ sqp*
shows *quote-reach P (quote-gross P sqp) ≤ sqp*
⟨*proof*⟩

lemma *quote-reach-strict-mono*:
  assumes *clmm-dsc P*
  and *nz-support (lq P) ≠ {}*
  and *0 ≤ y1*
  and *y1 < y2*
  and *y2 ≤ quote-gross P (grd-max P)*
  and *sqp = quote-reach P y1*
  and *sqp' = quote-reach P y2*
shows *sqp < sqp'*
⟨*proof*⟩

lemma *quote-reach-mono*:
  assumes *clmm-dsc P*
  and *nz-support (lq P) ≠ {}*
  and *0 ≤ y1*
  and *y1 ≤ y2*
  and *y2 ≤ quote-gross P (grd-max P)*
  and *sqp = quote-reach P y1*
  and *sqp' = quote-reach P y2*
shows *sqp ≤ sqp'*
⟨*proof*⟩

lemma *grd-max-quote-reach*:
  assumes *clmm-dsc P*
  and *nz-support (lq P) ≠ {}*
shows *quote-reach P (quote-gross P (grd-max P)) = grd-max P*
⟨*proof*⟩

lemma *quote-reach-gt*:
  assumes *clmm-dsc P*
  and *nz-support (lq P) ≠ {}*

and $0 < y$
and $y + \textit{quote-gross}\ P\ \textit{sqp} \leq \textit{quote-gross}\ P\ (\textit{grd-max}\ P)$
and $\textit{sqp}' = \textit{quote-reach}\ P\ (y + \textit{quote-gross}\ P\ \textit{sqp})$
shows $\textit{sqp} < \textit{sqp}'$
⟨*proof*⟩

lemma *lt-quote-gross-imp-up-price*:
assumes *clmm-dsc P*
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
and $0 < y$
and $y \leq \textit{quote-gross}\ P\ (\textit{grd-max}\ P)$
and $\textit{quote-gross}\ P\ \textit{sqp} < y$
and $\textit{sqp}' = \textit{quote-reach}\ P\ y$
shows $\textit{sqp} < \textit{sqp}'$
⟨*proof*⟩

lemma *quote-reach-add-gt*:
assumes *clmm-dsc P*
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
and $0 < y$
and $y + \textit{quote-gross}\ P\ \textit{sqp} \leq \textit{quote-gross}\ P\ (\textit{grd-max}\ P)$
and $\textit{sqp}' = \textit{quote-reach}\ P\ (y + \textit{quote-gross}\ P\ \textit{sqp})$
shows $\textit{quote-gross}\ P\ \textit{sqp} < \textit{quote-gross}\ P\ \textit{sqp}'$
⟨*proof*⟩

lemma *quote-reach-leq-grd-max*:
assumes *clmm-dsc P*
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
and $0 \leq y$
and $y \leq \textit{quote-gross}\ P\ (\textit{grd-max}\ P)$
and $\textit{sqp} = \textit{quote-reach}\ P\ y$
shows $\textit{sqp} \leq \textit{grd-max}\ P$
⟨*proof*⟩

lemma *quote-gross-grd-max-ge*:
assumes *clmm-dsc P*
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
and $\textit{grd-max}\ P < \textit{sqp}$
shows $\textit{quote-gross}\ P\ \textit{sqp} = \textit{quote-gross}\ P\ (\textit{grd-max}\ P)$
⟨*proof*⟩

lemma *quote-gross-grd-max-max*:
assumes *clmm-dsc P*
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
shows $\textit{quote-gross}\ P\ \textit{sqp} \leq \textit{quote-gross}\ P\ (\textit{grd-max}\ P)$
⟨*proof*⟩

lemma *gross-grd-max-max'*:
assumes *clmm-dsc P*

and *nz-support* (*lq P*) ≠ {}
and *sqp* < *grd-max P*
shows *quote-gross P sqp* < *quote-gross P* (*grd-max P*)
⟨*proof*⟩

lemma *quote-reach-le-gross*:
assumes *clmm-dsc P*
and *0* < *y*
and *0* < *sqp*
and *y* ≤ *quote-gross P sqp*
and *sqp* ≤ *grd-max P*
and *sqp′* = *quote-reach P y*
and *nz-support* (*lq P*) ≠ {}
shows *sqp′* ≤ *sqp*
⟨*proof*⟩

lemma *quote-net-diff-eq*:
assumes *clmm-dsc P*
and *L* = *lq P*
and *j* = *lower-tick P sqp*
and *k* = *lower-tick P sqp′*
and *0* < *sqp*
and *sqp* ≤ *sqp′*
and *j* < *k*
shows *quote-net P sqp′* − *quote-net P sqp* = *L k* ∗ (*sqp′* − *grd P k*) +
sum (λ *i*. *L i* ∗ (*grd P* (*i*+*1*) − *grd P i*)) {*i*. *L i* ≠ *0* ∧ *j* <*i* ∧ *i* < *k*} +
L j ∗ (*grd P* (*j*+*1*) − *sqp*)
⟨*proof*⟩

lemma *quote-net-diff-eq′*:
assumes *clmm-dsc P*
and *L* = *lq P*
and *j* = *lower-tick P sqp*
and *j* = *lower-tick P sqp′*
and *0* < *sqp*
and *sqp* ≤ *sqp′*
shows *quote-net P sqp′* − *quote-net P sqp* = *L j* ∗ (*sqp′* − *sqp*)
⟨*proof*⟩

4.3 Base token addition and withdrawal in a CLMM

lemma (**in** *finite-nz-support*) *gen-base-sum*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *j* = *lower-tick P sqp*
shows *gen-base* (*grd P*) *L sqp* =
L j ∗ (*inverse sqp* − *inverse* (*grd P* (*j*+*1*))) +
sum (λ*i*. *L i* ∗ (*inverse* (*grd P i*) − *inverse* (*grd P* (*i*+*1*))))
{*i*. *L i* ≠ *0* ∧ *j* < *i*}

⟨*proof*⟩

lemma (**in** *finite-nz-support*) *gen-base-grd-max*:
assumes *clmm-dsc P*
and *nz-support L* ≠ {}
and *nz-support L* = *nz-support* (*lq P*)
shows *gen-base* (*grd P*) *L* (*grd-max P*) = *0*
⟨*proof*⟩

lemma *base-gross-sum*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *L* = *gross-fct* (*lq P*) (*fee P*)
and *j* = *lower-tick P sqp*
shows *base-gross P sqp* =
L j ∗ (*inverse sqp* − *inverse* (*grd P* (*j*+*1*))) +
sum (λ*i*. *L i* ∗ (*inverse* (*grd P i*) − *inverse* (*grd P* (*i*+*1*))))
{*i*. *L i* ≠ *0* ∧ *j* < *i*}
⟨*proof*⟩

lemma *clmm-base-gross-grd-max*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
shows *base-gross P* (*grd-max P*) = *0* ⟨*proof*⟩

lemma *liq-base-reach-max*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
shows *base-reach P 0* = *grd-max P*
⟨*proof*⟩

lemma *base-net-sum*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *L* = *lq P*
and *j* = *lower-tick P sqp*
shows *base-net P sqp* =
L j ∗ (*inverse sqp* − *inverse* (*grd P* (*j*+*1*))) +
sum (λ*i*. *L i* ∗ (*inverse* (*grd P i*) − *inverse* (*grd P* (*i*+*1*))))
{*i*. *L i* ≠ *0* ∧ *j* < *i*}
⟨*proof*⟩

definition *gen-base-diff* **where**
gen-base-diff P L sqp sqp′ = *gen-base* (*grd P*) *L sqp* − *gen-base* (*grd P*) *L sqp*′

lemma (**in** *finite-nz-support*) *gen-base-diff-eq*:
assumes *clmm-dsc P*
and *j* = *lower-tick P sqp*
and *k* = *lower-tick P sqp*′

and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
shows *gen-base-diff P L sqp sqp'* =
$L\ j * (inverse\ sqp - inverse\ (grd\ P\ (j+1))) +$
$sum\ (\lambda\ i.\ L\ i * (inverse\ (grd\ P\ i) - inverse\ (grd\ P\ (i+1))))$
$\{i.\ L\ i \neq 0 \wedge j < i \wedge i < k\} +$
$L\ k * (inverse\ (grd\ P\ k) - inverse\ sqp')$
⟨*proof*⟩

lemma (**in** *finite-nz-support*) *gen-base-diff-eq'*:
assumes *clmm-dsc P*
and $j = lower\text{-}tick\ P\ sqp$
and $j = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
shows $gen\text{-}base\text{-}diff\ P\ L\ sqp\ sqp' = L\ j * (inverse\ sqp - inverse\ sqp')$
⟨*proof*⟩

lemma *lower-tick-lt-grd-min*:
assumes *clmm-dsc P*
and $sqp < grd\text{-}min\ P$
and $0 < sqp$
and $j = lower\text{-}tick\ P\ sqp$
shows $j < idx\text{-}min\ (lq\ P)$
⟨*proof*⟩

lemma (**in** *finite-nz-support*) *gen-base-grd-min-le*:
assumes *clmm-dsc P*
and $nz\text{-}support\ L = nz\text{-}support\ (lq\ P)$
and $sqp < grd\text{-}min\ P$
and $0 < sqp$
shows $gen\text{-}base\ (grd\ P)\ L\ sqp = gen\text{-}base\ (grd\ P)\ L\ (grd\text{-}min\ P)$
⟨*proof*⟩

lemma *base-net-grd-min-lt*:
assumes *clmm-dsc P*
and $sqp < grd\text{-}min\ P$
and $0 < sqp$
shows $base\text{-}net\ P\ sqp = base\text{-}net\ P\ (grd\text{-}min\ P)$
⟨*proof*⟩

lemma *base-net-grd-min-le*:
assumes *clmm-dsc P*
and $sqp \leq grd\text{-}min\ P$
and $0 < sqp$
shows $base\text{-}net\ P\ sqp = base\text{-}net\ P\ (grd\text{-}min\ P)$
⟨*proof*⟩

lemma *base-gross-diff-eq*:
assumes *clmm-dsc P*
and $L = gross\text{-}fct\ (lq\ P)\ (fee\ P)$
and $j = lower\text{-}tick\ P\ sqp$
and $k = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
shows $base\text{-}gross\ P\ sqp - base\text{-}gross\ P\ sqp' =$
$L\ j * (inverse\ sqp - inverse\ (grd\ P\ (j+1))) +$
$sum\ (\lambda\ i.\ L\ i * (inverse\ (grd\ P\ i) - inverse\ (grd\ P\ (i+1))))$
$\{i.\ L\ i \neq 0 \wedge j < i \wedge i < k\} +$
$L\ k * (inverse\ (grd\ P\ k) - inverse\ sqp')$
⟨*proof*⟩

lemma *base-gross-diff-eq'*:
assumes *clmm-dsc P*
and $L = gross\text{-}fct\ (lq\ P)\ (fee\ P)$
and $j = lower\text{-}tick\ P\ sqp$
and $j = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
shows $base\text{-}gross\ P\ sqp - base\text{-}gross\ P\ sqp' = L\ j * (inverse\ sqp - inverse\ sqp')$
⟨*proof*⟩

lemma *base-net-diff-eq*:
assumes *clmm-dsc P*
and $L = lq\ P$
and $j = lower\text{-}tick\ P\ sqp$
and $k = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
shows $base\text{-}net\ P\ sqp - base\text{-}net\ P\ sqp' =$
$L\ j * (inverse\ sqp - inverse\ (grd\ P\ (j+1))) +$
$sum\ (\lambda\ i.\ L\ i * (inverse\ (grd\ P\ i) - inverse\ (grd\ P\ (i+1))))$
$\{i.\ L\ i \neq 0 \wedge j < i \wedge i < k\} +$
$L\ k * (inverse\ (grd\ P\ k) - inverse\ sqp')$
⟨*proof*⟩

lemma *base-net-diff-eq'*:
assumes *clmm-dsc P*
and $L = lq\ P$
and $j = lower\text{-}tick\ P\ sqp$
and $j = lower\text{-}tick\ P\ sqp'$
and $0 < sqp$
and $sqp \leq sqp'$
shows $base\text{-}net\ P\ sqp - base\text{-}net\ P\ sqp' = L\ j * (inverse\ sqp - inverse\ sqp')$
⟨*proof*⟩

lemma *cst-fee-base-gross-net-tick-eq*:
assumes *clmm-dsc P*
and $\bigwedge i$. *fee P i* = *phi*
and *j* = *lower-tick P sqp*
and *j* = *lower-tick P sqp'*
and *0* < *sqp*
and *sqp* ≤ *sqp'*
shows *base-net P sqp − base-net P sqp'* =
(*1 − phi*) ∗ (*base-gross P sqp − base-gross P sqp'*)
⟨*proof*⟩

lemma *cst-fee-base-gross-net-tick-lt*:
assumes *clmm-dsc P*
and $\bigwedge i$. *fee P i* = *phi*
and *j* = *lower-tick P sqp*
and *k* = *lower-tick P sqp'*
and *0* < *sqp*
and *sqp* ≤ *sqp'*
and *j* < *k*
shows *base-net P sqp − base-net P sqp'* =
(*1 − phi*) ∗ (*base-gross P sqp − base-gross P sqp'*)
⟨*proof*⟩

lemma *cst-fee-base-gross-net*:
assumes *clmm-dsc P*
and $\bigwedge i$. *fee P i* = *phi*
and *0* < *sqp*
and *sqp* ≤ *sqp'*
shows *base-net P sqp − base-net P sqp'* =
(*1 − phi*) ∗ (*base-gross P sqp − base-gross P sqp'*)
⟨*proof*⟩

lemma *base-net-eq-only-if*:
assumes *clmm-dsc P*
and *j* = *lower-tick P sqp*
and *k* = *lower-tick P sqp'*
and *0* < *sqp*
and *sqp* ≤ *sqp'*
and *j* < *k*
and *quote-gross P sqp'* = *quote-gross P sqp*
shows *base-net P sqp'* = *base-net P sqp*
⟨*proof*⟩

lemma *base-net-eq-only-if'*:
assumes *clmm-dsc P*
and *L* = *lq P*
and *j* = *lower-tick P sqp*
and *j* = *lower-tick P sqp'*

and $0 < sqp$
and $sqp \leq sqp'$
and $quote\text{-}gross\ P\ sqp = quote\text{-}gross\ P\ sqp'$
shows $base\text{-}net\ P\ sqp = base\text{-}net\ P\ sqp'$
⟨*proof*⟩

lemma *quote-gross-equiv-base-net*:
assumes *clmm-dsc P*
and $0 < sqp$
and $sqp \leq sqp'$
and $quote\text{-}gross\ P\ sqp = quote\text{-}gross\ P\ sqp'$
shows $base\text{-}net\ P\ sqp = base\text{-}net\ P\ sqp'$
⟨*proof*⟩

lemma *quote-gross-equiv-base-net′*:
assumes *clmm-dsc P*
and $0 < sqp$
and $0 < sqp'$
and $quote\text{-}gross\ P\ sqp = quote\text{-}gross\ P\ sqp'$
shows $base\text{-}net\ P\ sqp = base\text{-}net\ P\ sqp'$
⟨*proof*⟩

lemma (**in** *finite-nz-support*) *gen-quote-le-badd*:
assumes *clmm-dsc P*
and $\bigwedge i.\ 0 \leq L\ i$
and $0 < sqp$
and $sqp \leq sqp'$
shows $gen\text{-}quote\text{-}diff\ P\ L\ sqp\ sqp'/(sqp' * sqp') \leq gen\text{-}base\text{-}diff\ P\ L\ sqp\ sqp'$
⟨*proof*⟩

lemma (**in** *finite-nz-support*) *gen-base-le-qadd*:
assumes *clmm-dsc P*
and $\bigwedge i.\ 0 \leq L\ i$
and $0 < sqp$
and $sqp \leq sqp'$
shows $gen\text{-}base\text{-}diff\ P\ L\ sqp\ sqp' \leq gen\text{-}quote\text{-}diff\ P\ L\ sqp\ sqp'/(sqp * sqp)$
⟨*proof*⟩

lemma *quote-swap-grd-min-zero*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $grd\text{-}min\ P \leq sqp$
and $sqp \leq grd\text{-}max\ P$
shows $quote\text{-}swap\ P\ sqp\ 0 = 0$
⟨*proof*⟩

lemma *quote-swap-zero*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$

and $0 < sqp$
and $sqp \leq grd\text{-}max\ P$
shows $quote\text{-}swap\ P\ sqp\ 0 = 0$
⟨*proof*⟩

lemma *quote-swap-grd-min-zero'*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $grd\text{-}min\ P \leq sqp$
and $quote\text{-}gross\ P\ sqp \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
shows $quote\text{-}swap\ P\ sqp\ 0 = 0$
⟨*proof*⟩

lemma *quote-swap-zero'*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $0 < sqp$
and $quote\text{-}gross\ P\ sqp \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
shows $quote\text{-}swap\ P\ sqp\ 0 = 0$
⟨*proof*⟩

lemma *quote-swap-grd-min*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $sqp < grd\text{-}min\ P$
and $0 < sqp$
shows $quote\text{-}swap\ P\ sqp\ y = quote\text{-}swap\ P\ (grd\text{-}min\ P)\ y$
⟨*proof*⟩

lemma *quote-reach-gross-base-net*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $0 < quote\text{-}gross\ P\ sqp$
and $sqp' = quote\text{-}reach\ P\ (quote\text{-}gross\ P\ sqp)$
shows $base\text{-}net\ P\ sqp' = base\text{-}net\ P\ sqp$
⟨*proof*⟩

lemma *quote-reach-base-net*:
assumes *clmm-dsc P*
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $0 < sqp$
and $sqp' = quote\text{-}reach\ P\ (quote\text{-}gross\ P\ sqp)$
shows $base\text{-}net\ P\ sqp' = base\text{-}net\ P\ sqp$
⟨*proof*⟩

lemma *base-le-quote-gross*:
assumes *clmm-dsc P'*
and $0 < sqp$
and $sqp \leq sqp'$

shows *base-gross P′ sqp − base-gross P′ sqp′ ≤*
(quote-gross P′ sqp′ − quote-gross P′ sqp)/(sqp ∗ sqp)
⟨*proof*⟩

lemma *quote-le-base-gross*:
assumes *clmm-dsc P′*
and *0 < sqp*
and *sqp ≤ sqp′*
shows *(quote-gross P′ sqp′ − quote-gross P′ sqp)/(sqp′ ∗ sqp′) ≤*
base-gross P′ sqp − base-gross P′ sqp′
⟨*proof*⟩

lemma *base-net-quote-ubound*:
assumes *clmm-dsc P′*
and *⋀i. fee P′ i = phi*
and *phi < 1*
and *0 < sqp*
and *sqp ≤ sqp′*
shows *base-net P′ sqp − base-net P′ sqp′ ≤*
(1 − phi) ∗ (quote-gross P′ sqp′ − quote-gross P′ sqp)/(sqp ∗ sqp)
⟨*proof*⟩

lemma *base-net-quote-lbound*:
assumes *clmm-dsc P′*
and *⋀i. fee P′ i = phi*
and *0 < sqp*
and *sqp ≤ sqp′*
shows *(1 − phi) ∗ (quote-gross P′ sqp′ − quote-gross P′ sqp)/(sqp′ ∗ sqp′) ≤*
base-net P′ sqp − base-net P′ sqp′
⟨*proof*⟩

4.4 Market depth and slippage for finer CLMMs

4.4.1 Finer pools

locale *finer-clmm =*
fixes *P1 P2*
assumes *abs1*: *clmm-dsc P1* **and** *abs2*: *clmm-dsc P2*
and *finer*: *finer-pool P1 P2*

sublocale *finer-clmm ⊆ finer-two-span-finite-liq*
⟨*proof*⟩

context *finer-clmm*
begin

lemma *finer-base-net-eq*:
shows *base-net P1 = base-net P2*
⟨*proof*⟩

lemma *finer-quote-net-eq*:
shows *quote-net P1 = quote-net P2*
⟨*proof*⟩

lemma *finer-base-gross-eq*:
shows *base-gross P1 = base-gross P2*
⟨*proof*⟩

lemma *finer-quote-gross-eq*:
shows *quote-gross P1 = quote-gross P2*
⟨*proof*⟩

lemma *finer-mkt-depth*:
shows *mkt-depth P1 = mkt-depth P2*
⟨*proof*⟩

end

4.4.2 Finer CLMMs with nonzero liquidity

locale *finer-clmm-ne = finer-clmm +*
assumes *nonempty-liq*: *nz-support* (*lq P1*) ≠ {}

context *finer-clmm-ne*
begin

lemma *id-max-Max-eq*:
assumes *i1 = idx-max* (*lq P1*)
and *k2 = pool-coarse-idx P1 P2 i1*
shows *i1 = Max* (*encompassed* (*grd P1*) (*grd P2*) *k2*)
⟨*proof*⟩

lemma *id-min-Min-eq*:
assumes *i1 = idx-min* (*lq P1*)
and *k2 = pool-coarse-idx P1 P2 i1*
shows *i1 = Min* (*encompassed* (*grd P1*) (*grd P2*) *k2*)
⟨*proof*⟩

lemma *idx-max-Suc-grd-eq*:
assumes *i1 = idx-max* (*lq P1*)
and *k2 = pool-coarse-idx P1 P2 i1*
shows *grd P1* (*i1 + 1*) = *grd P2* (*k2 + 1*)
⟨*proof*⟩

lemma *idx-min-grd-eq*:
assumes *i1 = idx-min* (*lq P1*)
and *k2 = pool-coarse-idx P1 P2 i1*
shows *grd P1 i1 = grd P2 k2*
⟨*proof*⟩

lemma *abs-finer-idx-max-coarse*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *finer-pool P1 P2*
and *nz-support* (*lq P1*) ≠ {}
and *i1* = *idx-max* (*lq P1*)
and *k2* = *pool-coarse-idx P1 P2 i1*
shows *k2* = *idx-max* (*lq P2*)
⟨*proof*⟩

lemma *abs-finer-idx-min-coarse*:
assumes *i1* = *idx-min* (*lq P1*)
and *k2* = *pool-coarse-idx P1 P2 i1*
shows *k2* = *idx-min* (*lq P2*)
⟨*proof*⟩

lemma *abs-finer-idx-max-img-eq*:
shows *grd-max P1* = *grd-max P2*
⟨*proof*⟩

lemma *abs-finer-idx-min-img-eq*:
shows *grd-min P1* = *grd-min P2*
⟨*proof*⟩

lemma *finer-base-reach-eq*:
shows *base-reach P1* = *base-reach P2* ⟨*proof*⟩

lemma *finer-quote-reach-eq*:
shows *quote-reach P1* = *quote-reach P2* ⟨*proof*⟩

lemma *finer-base-slippage*:
shows *base-slippage P1* = *base-slippage P2*
⟨*proof*⟩

lemma *finer-quote-slippage*:
shows *quote-slippage P1* = *quote-slippage P2*
⟨*proof*⟩

end

5 Inequalities related to fees

context *finite-liq-pool*
begin

lemma *gross-fct-le*:
assumes *0* ≤ *f i*
and *phi i* ≤ *phi′ i*

and *phi′ i < 1*
shows *gross-fct f phi i ≤ gross-fct f phi′ i*
⟨*proof*⟩

lemma *gross-fct-lt*:
assumes *0 < f i*
and *phi i < phi′ i*
and *phi′ i < 1*
shows *gross-fct f phi i < gross-fct f phi′ i*
⟨*proof*⟩

lemma *fee-diff-same-base-net*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and $I = \{k.\ L\ k \neq 0 \wedge j \leq k\}$
and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′ {k. j ≤ k}*
and *0 < sqp*
and *j = lower-tick P sqp*
and *L = lq P*
and *lower-tick P sqp = lower-tick P′ sqp*
shows *base-net P sqp = base-net P′ sqp*
⟨*proof*⟩

lemma *fee-diff-le-imp-quote-gross*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and $\{k.\ L\ k \neq 0 \wedge k \leq j\} \subseteq I$
and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′ {k. k ≤ j}*
and *0 < sqp*
and *j = lower-tick P sqp*
and *L = gross-fct (lq P) (fee P)*
and $\bigwedge i.\ i \in I \Longrightarrow$ *fee P i ≤ fee P′ i*
and *lower-tick P sqp = lower-tick P′ sqp*
shows *quote-gross P sqp ≤ quote-gross P′ sqp*
⟨*proof*⟩

lemma *fee-diff-le-imp-quote-gross-mono*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and $\{k.\ L\ k \neq 0 \wedge k \leq j\} \subseteq I$
and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′ {k. k ≤ j}*
and *0 < sqp*
and *j = lower-tick P sqp*
and *L = gross-fct (lq P) (fee P)*
and $\bigwedge i.\ i \in I \Longrightarrow$ *fee P i ≤ fee P′ i*
and *lower-tick P sqp = lower-tick P′ sqp*

and $sqp \leq sqp'$
shows *quote-gross* P $sqp \leq$ *quote-gross* P' sqp'
⟨*proof*⟩

lemma *fee-diff-quote-diff-expand*:
assumes *clmm-dsc* P
and *clmm-dsc* P'
and $L =$ *gross-fct* (*lq* P) (*fee* P)
and $j =$ *lower-tick* P *sqp*
and $k =$ *lower-tick* P sqp'
and $0 < sqp$
and $sqp \leq sqp'$
and $j < k$
and $\{m.\ L\ m \neq 0 \wedge j \leq m \wedge m \leq k\} \subseteq I$
and *fee-diff-on* P P' I
and *same-nz-liq-on* P P' $\{m.\ j \leq m \wedge m \leq k{+}1\}$
and $\bigwedge i.\ i \in I \Longrightarrow$ *fee* P $i \leq$ *fee* P' i
and *lower-tick* P *sqp* = *lower-tick* P' *sqp*
and *lower-tick* P sqp' = *lower-tick* P' sqp'
shows *quote-gross* P $sqp' -$ *quote-gross* P *sqp* $\leq$ *quote-gross* P' $sqp' -$ *quote-gross* P' *sqp*
⟨*proof*⟩

lemma *fee-diff-quote-diff-expand'*:
assumes *clmm-dsc* P
and *clmm-dsc* P'
and $L =$ *gross-fct* (*lq* P) (*fee* P)
and $j =$ *lower-tick* P *sqp*
and $j =$ *lower-tick* P sqp'
and $0 < sqp$
and $sqp \leq sqp'$
and $L\ j \neq 0 \longrightarrow j \in I$
and *fee-diff-on* P P' I
and $\bigwedge i.\ i \in I \Longrightarrow$ *fee* P $i \leq$ *fee* P' i
and *lower-tick* P *sqp* = *lower-tick* P' *sqp*
and *lower-tick* P sqp' = *lower-tick* P' sqp'
shows *quote-gross* P $sqp' -$ *quote-gross* P *sqp* $\leq$ *quote-gross* P' $sqp' -$ *quote-gross* P' *sqp*
⟨*proof*⟩

lemma *fee-diff-quote-diff-le*:
assumes *clmm-dsc* P
and *clmm-dsc* P'
and $L =$ *gross-fct* (*lq* P) (*fee* P)
and $j =$ *lower-tick* P *sqp*
and $k =$ *lower-tick* P sqp'
and $0 < sqp$
and $sqp \leq sqp'$
and $\{m.\ L\ m \neq 0 \wedge j \leq m \wedge m \leq k\} \subseteq I$

and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′ {m. j ≤ m ∧ m ≤ k+1}*
and $\bigwedge i.\ i \in I \Longrightarrow$ *fee P i ≤ fee P′ i*
and *lower-tick P sqp = lower-tick P′ sqp*
and *lower-tick P sqp′ = lower-tick P′ sqp′*
shows *quote-gross P sqp′ − quote-gross P sqp ≤ quote-gross P′ sqp′ − quote-gross P′ sqp*
⟨*proof*⟩

lemma *same-nz-liq-on-nz-support*:
assumes *i ∈ I*
and *lq P i ≠ 0*
and *same-nz-liq-on P P′ I*
shows *nz-support (lq P′) ≠ {}*
⟨*proof*⟩

lemma *same-nz-liq-on-idx-max*:
assumes *finite-liq P′*
and *nz-support (lq P) ≠ {}*
and *I = {idx-min (lq P) .. idx-max (lq P) + 1}*
and *same-nz-liq-on P P′ I*
shows *idx-max (lq P) ≤ idx-max (lq P′)*
⟨*proof*⟩

lemma *same-nz-liq-on-grd-max*:
assumes *finite-liq P′*
and *mono (grd P′)*
and *nz-support (lq P) ≠ {}*
and *I = {idx-min (lq P) .. idx-max (lq P) + 1}*
and *same-nz-liq-on P P′ I*
shows *grd-max P ≤ grd-max P′*
⟨*proof*⟩

lemma *same-nz-liq-on-lower-tick*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *same-nz-liq-on P P′ {i. i ≤ j+1}*
and *0 < sqp*
and *lower-tick P sqp ≤ j*
shows *lower-tick P′ sqp = lower-tick P sqp*
⟨*proof*⟩

lemma *same-nz-liq-on-lower-tick′*:
assumes *clmm-dsc P′*
and *same-nz-liq-on P P′ {i. i ≤ j}*
and *grd P j = sqp*
shows *lower-tick P′ sqp = j*
⟨*proof*⟩

lemma *fee-diff-le-grd-max*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and {*idx-min* (*lq P*) .. *idx-max* (*lq P*) + *1*} ⊆ *I*
and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′* {*k*. *k* ≤ *idx-max* (*lq P*) + *1*}
and ⋀*i*. *i* ∈ *I* ⟹ *fee P i* ≤ *fee P′ i*
shows *quote-gross P* (*grd-max P*) ≤ *quote-gross P′* (*grd-max P*)
⟨*proof*⟩

lemma *fee-diff-le-grd-max′*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and {*idx-min* (*lq P*) .. *idx-max* (*lq P*) + *1*} ⊆ *I*
and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′* {*k*. *k* ≤ *idx-max* (*lq P*) + *1*}
and ⋀*i*. *i* ∈ *I* ⟹ *fee P i* ≤ *fee P′ i*
shows *quote-gross P* (*grd-max P*) ≤ *quote-gross P′* (*grd-max P′*)
⟨*proof*⟩

lemma *fee-diff-le-imp-quote-reach*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and {*idx-min* (*lq P*) .. *idx-max* (*lq P*) + *1*} ⊆ *I*
and *fee-diff-on P P′ I*
and *same-nz-liq-on P P′* {*i*. *i* ≤ *idx-max* (*lq P*) + *1*}
and ⋀*i*. *i* ∈ *I* ⟹ *fee P i* ≤ *fee P′ i*
and *0* < *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
shows *quote-reach P′ y* ≤ *quote-reach P y*
⟨*proof*⟩

lemma *same-nz-liq-on-if-simil*:
assumes *grd P* = *grd P′*
and *nz-support* (*lq P*) = *nz-support* (*lq P′*)
shows *same-nz-liq-on P P′ I*
⟨*proof*⟩

lemma *fee-diff-simil-base-net*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and {*idx-min* (*lq P*) .. *idx-max* (*lq P*) + *1*} ⊆ *I*
and *fee-diff-on P P′ I*
and *nz-support* (*lq P*) = *nz-support* (*lq P′*)
and *grd P* = *grd P′*

and *grd-min P* ≤ *sqp*
and *sqp* ≤ *grd-max P*
shows *base-net P sqp* = *base-net P′ sqp*
⟨*proof*⟩

lemma *fee-diff-le-price-cmp*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and {*idx-min* (*lq P*) .. *idx-max* (*lq P*) + *1*} ⊆ *I*
and *fee-diff-on P P′ I*
and *nz-support* (*lq P*) = *nz-support* (*lq P′*)
and *grd P* = *grd P′*
and ⋀*i*. *i* ∈ *I* ⟹ *fee P i* ≤ *fee P′ i*
and *0* < *y*
and *y* + *quote-gross P sqp* ≤ *quote-gross P* (*grd-max P*)
and *grd-min P* ≤ *sqp*
and *sqp1* = *quote-reach P* (*y* + *quote-gross P sqp*)
and *sqp2* = *quote-reach P′* (*y* + *quote-gross P′ sqp*)
shows *sqp2* ≤ *sqp1*
⟨*proof*⟩

lemma *fee-diff-le-imp-quote-swap*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and {*idx-min* (*lq P*) .. *idx-max* (*lq P*) + *1*} ⊆ *I*
and *fee-diff-on P P′ I*
and *nz-support* (*lq P*) = *nz-support* (*lq P′*)
and *grd P* = *grd P′*
and ⋀*i*. *i* ∈ *I* ⟹ *fee P i* ≤ *fee P′ i*
and *0* < *y*
and *y* + *quote-gross P sqp* ≤ *quote-gross P* (*grd-max P*)
and *grd-min P* ≤ *sqp*
shows *quote-swap P′ sqp y* ≤ *quote-swap P sqp y*
⟨*proof*⟩

lemma *fee-ge-quote-swap-le*:
assumes *clmm-dsc P*
and *clmm-dsc P′*
and *nz-support* (*lq P*) ≠ {}
and *grd P* = *grd P′*
and *lq P* = *lq P′*
and ⋀*i*. *fee P i* ≤ *fee P′ i*
and *0* ≤ *y*
and *0* < *sqp*
and *y* + *quote-gross P sqp* ≤ *quote-gross P* (*grd-max P*)
shows *quote-swap P′ sqp y* ≤ *quote-swap P sqp y*
⟨*proof*⟩

end

end
theory *CLMM-Transformation* **imports** *CLMM-Description*

begin

6 CLMM transformations

6.1 CLMM pool refinement

Given a pool P and a (square root) price π, the refinement operation consists in defining a new grid (if necessary) in such a way that π is one of the bounds on the grid.

definition *refine* **where**
refine P sqp = (*let i* = *lower-tick P sqp in*
(*if* (*grd P i* = *sqp*) *then P else*
(*wedge* (*grd P*) *i sqp*, *wedge* (*lq P*) *i* (*lq P i*), *wedge* (*fee P*) *i* (*fee P i*))))

lemma *refine-eq*:
assumes *i* = *lower-tick P sqp*
and *grd P i* = *sqp*
shows *refine P sqp* = *P* ⟨*proof*⟩

lemma *refine-lq*:
assumes *i* = *lower-tick P sqp*
and *grd P i* ≠ *sqp*
and *P′* = *refine P sqp*
shows *lq P′* = *wedge* (*lq P*) *i* (*lq P i*)
⟨*proof*⟩

lemma *refine-fee*:
assumes *i* = *lower-tick P sqp*
and *grd P i* ≠ *sqp*
and *P′* = *refine P sqp*
shows *fee P′* = *wedge* (*fee P*) *i* (*fee P i*)
⟨*proof*⟩

lemma *refine-grd*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *grd P i* ≠ *sqp*
shows *grd P′* = *wedge* (*grd P*) *i sqp*
⟨*proof*⟩

lemma *refine-grd-cong*:
assumes *P1* = *refine P sqp*
and *P2* = *refine P′ sqp*
and *grd P* = *grd P′*
shows *grd P1* = *grd P2*
⟨*proof*⟩

lemma *refine-grd-arg-le*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *j* ≤ *i*
shows *grd P′ j* = *grd P j*
⟨*proof*⟩

lemma *refine-grd-arg-gt*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
and *i* < *j*
shows *grd P′* (*j*+*1*) = *grd P j*
⟨*proof*⟩

lemma *refine-grd-arg-Suc*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
shows *grd P′* (*i*+*1*) = *sqp*
⟨*proof*⟩

lemma *refine-fee-arg-le*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *j* ≤ *i*
shows *fee P′ j* = *fee P j*
⟨*proof*⟩

lemma *refine-fee-arg-gt*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
and *i* < *j*
shows *fee P′* (*j*+*1*) = *fee P j*
⟨*proof*⟩

lemma *refine-fee-arg-Suc*:
assumes *i* = *lower-tick P sqp*
and *P′* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*

shows *fee* P' $(i+1)$ = *fee* P i
⟨*proof*⟩

lemma *refine-lq-arg-le*:
  assumes i = *lower-tick* P *sqp*
  and P' = *refine* P *sqp*
  and *grd* P $i \neq$ *sqp*
  and $j \leq i$
shows *lq* P' j = *lq* P j
⟨*proof*⟩

lemma *refine-lq-arg-gt*:
  assumes i = *lower-tick* P *sqp*
  and P' = *refine* P *sqp*
  and *grd* P $i \neq$ *sqp*
  and $i < j$
shows *lq* P' $(j+1)$ = *lq* P j
⟨*proof*⟩

lemma *refine-lq-arg-Suc*:
  assumes i = *lower-tick* P *sqp*
  and P' = *refine* P *sqp*
  and *grd* P $i \neq$ *sqp*
shows *lq* P' $(i+1)$ = *lq* P i
⟨*proof*⟩

lemma *refine-on-grd*:
  assumes *clmm-dsc* P
  and *grd* P i = *sqp*
  shows *refine* P *sqp* = P
⟨*proof*⟩

lemma *refine-encomp-at-grd*:
  assumes *clmm-dsc* P
  and P' = *refine* P *sqp*
  and *grd* P (*lower-tick* P *sqp*) = *sqp*
shows *encomp-at* (*grd* P') (*grd* P) j j
⟨*proof*⟩

lemma *refine-encomp-at-arg-le*:
  assumes *clmm-dsc* P
  and P' = *refine* P *sqp*
  and i = *lower-tick* P *sqp*
  and *grd* P $i \neq$ *sqp*
  and $j \leq i$
shows *encomp-at* (*grd* P') (*grd* P) j j
⟨*proof*⟩

lemma *refine-encomp-at-arg-ge-Suc*:

assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$
and $i = lower\text{-}tick\ P\ sqp$
and $grd\ P\ i \neq sqp$
and $i{+}1 \leq j$
and $0 < sqp$
shows $encomp\text{-}at\ (grd\ P')\ (grd\ P)\ j\ (j{-}1)$
⟨*proof*⟩

lemma *refine-finer-range*:
assumes *clmm-dsc P*
and $0 < sqp$
and $P' = refine\ P\ sqp$
shows $finer\text{-}range\ (grd\ P')\ (grd\ P)$
⟨*proof*⟩

lemma *refine-finite-liq*:
assumes *finite-liq P*
and $P' = refine\ P\ sqp$
shows *finite-liq P′*
⟨*proof*⟩

lemma *refine-clmm*:
assumes *clmm-dsc P*
and $0 < sqp$
and $P' = refine\ P\ sqp$
shows *clmm-dsc P′*
⟨*proof*⟩

lemma *refine-lower-tick-idx*:
assumes *clmm-dsc P*
and $0 < sqp$
and $i = lower\text{-}tick\ P\ sqp$
and $P' = refine\ P\ sqp$
and $grd\ P\ (lower\text{-}tick\ P\ sqp) \neq sqp$
shows $lower\text{-}tick\ P'\ sqp = i{+}1$
⟨*proof*⟩

lemma *refine-ge-lower-tick-eq*:
assumes *clmm-dsc P*
and $0 < sqp$
and $i = lower\text{-}tick\ P\ sqp'$
and $P' = refine\ P\ sqp$
and $grd\ P\ (lower\text{-}tick\ P\ sqp) \neq sqp$
and $sqp \leq sqp'$
and $lower\text{-}tick\ P\ sqp = lower\text{-}tick\ P\ sqp'$
shows $lower\text{-}tick\ P'\ sqp' = i{+}1$
⟨*proof*⟩

lemma *refine-ge-lower-tick-gt*:
assumes *clmm-dsc P*
and $0 < sqp$
and $sqp \leq sqp'$
and $i =$ *lower-tick P sqp'*
and $P' =$ *refine P sqp*
and *grd P* (*lower-tick P sqp*) $\neq sqp$
and *lower-tick P sqp* < *lower-tick P sqp'*
shows *lower-tick P' sqp'* $= i+1$
⟨*proof*⟩

lemma *refine-ge-lower-tick*:
assumes *clmm-dsc P*
and $0 < sqp$
and $sqp \leq sqp'$
and $i =$ *lower-tick P sqp'*
and $P' =$ *refine P sqp*
and *grd P* (*lower-tick P sqp*) $\neq sqp$
shows *lower-tick P' sqp'* $= i+1$
⟨*proof*⟩

lemma *refine-lower-tick*:
assumes *clmm-dsc P*
and $P' =$ *refine P sqp*
and $0 < sqp$
shows *grd P'* (*lower-tick P' sqp*) $= sqp$
⟨*proof*⟩

lemma *refine-finer-ranges*:
assumes *clmm-dsc P*
and $0 < sqp$
and $P' =$ *refine P sqp*
shows *finer-ranges* (*grd P'*) (*grd P*)
⟨*proof*⟩

lemma *refine-coarse-idx-grd*:
assumes *clmm-dsc P*
and $P' =$ *refine P sqp*
and *grd P* (*lower-tick P sqp*) $= sqp$
shows *coarse-idx* (*grd P'*) (*grd P*) $j = j$
⟨*proof*⟩

lemma *refine-coarse-idx-arg-le*:
assumes *clmm-dsc P*
and $P' =$ *refine P sqp*
and $i =$ *lower-tick P sqp*
and *grd P i* $\neq sqp$
and $j \leq i$
and $0 < sqp$

shows *coarse-idx* (*grd P'*) (*grd P*) *j* = *j*
⟨*proof*⟩

lemma *refine-coarse-idx-arg-gt*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *refine P sqp*
and *i* = *lower-tick P sqp*
and *grd P i* ≠ *sqp*
and *i+1* ≤ *j*
shows *coarse-idx* (*grd P'*) (*grd P*) *j* = *j−1*
⟨*proof*⟩

lemma *refine-lq-idx-neq*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
shows *lq P' j* = *lq P* (*pool-coarse-idx P' P j*)
⟨*proof*⟩

lemma *refine-fee-idx-neq*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
shows *fee P' j* = *fee P* (*pool-coarse-idx P' P j*)
⟨*proof*⟩

lemma *refine-cst-fees*:
assumes ⋀*i*. *fee P i* = *phi*
and *P'* = *refine P sqp*
shows ⋀*i*. *fee P' i* = *phi*
⟨*proof*⟩

lemma *refine-finer-neq*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *refine P sqp*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
shows *finer-pool P' P*
⟨*proof*⟩

lemma *refine-finer*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *refine P sqp*
shows *finer-pool P' P*
⟨*proof*⟩

lemma *refine-nz-lq-sub*:
assumes *clmm-dsc P*
and *0 < sqp*
and $P' = refine\ P\ sqp$
shows $(\lambda j.\ pool\text{-}coarse\text{-}idx\ P'\ P\ j)\ `\ nz\text{-}support\ (lq\ P') \subseteq$
nz-support (*lq P*)
⟨*proof*⟩

lemma *refine-nz-lq-ne*:
assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$
and $nz\text{-}support\ (lq\ P) \neq \{\}$
shows $nz\text{-}support\ (lq\ P') \neq \{\}$
⟨*proof*⟩

lemma *refine-nz-lq-emp*:
assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$
and $nz\text{-}support\ (lq\ P) = \{\}$
shows $nz\text{-}support\ (lq\ P') = \{\}$
⟨*proof*⟩

lemma *refine-idx-min-eq*:
assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$
and $idx\text{-}min\ (lq\ P) \leq lower\text{-}tick\ P\ sqp$
shows $idx\text{-}min\ (lq\ P') = idx\text{-}min\ (lq\ P)$
⟨*proof*⟩

lemma *refine-idx-min-Suc-eq*:
assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$
and $nz\text{-}support\ (lq\ P) \neq \{\}$
and $grd\ P\ (lower\text{-}tick\ P\ sqp) \neq sqp$
and $lower\text{-}tick\ P\ sqp < idx\text{-}min\ (lq\ P)$
shows $idx\text{-}min\ (lq\ P') = idx\text{-}min\ (lq\ P) + 1$
⟨*proof*⟩

lemma *refine-grd-min*:
assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$
and $nz\text{-}support\ (lq\ P) \neq \{\}$
shows $grd\text{-}min\ P = grd\text{-}min\ P'$
⟨*proof*⟩

lemma *refine-idx-max-eq*:
assumes *clmm-dsc P*
and $P' = refine\ P\ sqp$

and *idx-max* (*lq P*) < *lower-tick P sqp*
shows *idx-max* (*lq P'*) = *idx-max* (*lq P*)
⟨*proof*⟩

lemma *refine-idx-max-Suc-eq*:
assumes *clmm-dsc P*
and *P'* = *refine P sqp*
and *nz-support* (*lq P*) ≠ {}
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
and *lower-tick P sqp* ≤ *idx-max* (*lq P*)
shows *idx-max* (*lq P'*) = *idx-max* (*lq P*) + *1*
⟨*proof*⟩

lemma *refine-lower-tick-idx-max*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *refine P sqp*
and *nz-support* (*lq P*) ≠ {}
and *lower-tick P sqp* ≤ *idx-max* (*lq P*)
shows *lower-tick P' sqp* ≤ *idx-max* (*lq P'*)
⟨*proof*⟩

lemma *refine-grd-max*:
assumes *clmm-dsc P*
and *P'* = *refine P sqp*
and *nz-support* (*lq P*) ≠ {}
shows *grd-max P* = *grd-max P'*
⟨*proof*⟩

lemma *refine-quote-gross*:
assumes *clmm-dsc P*
and *P'* = *refine P sqp*
and *0* < *sqp*
shows *quote-gross P'* = *quote-gross P*
⟨*proof*⟩

lemma *refine-nonzero-liq*:
assumes *clmm-dsc P*
and *lower-tick P sqp* ≤ *i*
and *grd P* (*lower-tick P sqp*) ≠ *sqp*
and *P'* = *refine P sqp*
and *L* = *lq P*
and *L'* = *lq P'*
shows {*l*. *L' l* ≠ *0* ∧ *i*+*1* < *l*} = (λ*i*. *i* + *1*) ‘ {*k*. *L k* ≠ *0* ∧ *i* < *k*}
⟨*proof*⟩

lemma *refine-pool-base-net-grd-eq*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}

and $P' = \mathit{refine}\ P\ \mathit{sqp}$
and $0 < \mathit{sqp}$
and $\mathit{sqp} < \mathit{grd\text{-}max}\ P$
and $\mathit{grd}\ P\ (\mathit{lower\text{-}tick}\ P\ \mathit{sqp}) \neq \mathit{sqp}$
and $\mathit{sqp} \leq \mathit{sqp}'$
shows $\mathit{base\text{-}net}\ P'\ \mathit{sqp}' = \mathit{base\text{-}net}\ P\ \mathit{sqp}'$
⟨*proof*⟩

lemma *refine-base-net-eq*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and $P' = \mathit{refine}\ P\ \mathit{sqp}$
and $0 < \mathit{sqp}$
and $\mathit{sqp} < \mathit{grd\text{-}max}\ P$
and $\mathit{sqp} \leq \mathit{sqp}'$
shows $\mathit{base\text{-}net}\ P'\ \mathit{sqp}' = \mathit{base\text{-}net}\ P\ \mathit{sqp}'$
⟨*proof*⟩

6.2 CLMM pool restriction and slice

The restriction operation intuitively consists in deleting all the liquidity potentially available below the index provided as an argument.

definition *restrict-pool* **where**
restrict-pool i P =
(*grd P*,
(λ*j*. *if j* < *i then 0 else lq P j*),
(λ*j*. *fee P j*))

lemma *restrict-pool-grd*[*simp*]:
shows *grd* (*restrict-pool i P*) = *grd P*
⟨*proof*⟩

lemma *restrict-pool-lower-tick*:
assumes $P' = \mathit{restrict\text{-}pool}\ i\ P$
shows $\mathit{lower\text{-}tick}\ P\ \mathit{sqp} = \mathit{lower\text{-}tick}\ P'\ \mathit{sqp}$
⟨*proof*⟩

lemma *restrict-pool-lt*:
assumes $j < i$
shows *lq* (*restrict-pool i P*) *j* = *0 fee* (*restrict-pool i P*) *j* = *fee P j*
⟨*proof*⟩

lemma *restrict-pool-ge*:
assumes $i \leq j$
shows *lq* (*restrict-pool i P*) *j* = *lq P j*
fee (*restrict-pool i P*) *j* = *fee P j*
⟨*proof*⟩

lemma *restrict-pool-lq-sub*:

assumes $P' = \textit{restrict-pool}\ i\ P$
shows $\textit{nz-support}\ (\textit{lq}\ P') \subseteq \textit{nz-support}\ (\textit{lq}\ P)$
⟨*proof*⟩

lemma *restrict-pool-finite-liq*:
assumes *finite-liq P*
and $P' = \textit{restrict-pool}\ i\ P$
shows *finite-liq* P' ⟨*proof*⟩

lemma *restrict-pool-nz-liq*:
assumes *finite-liq P*
and $P' = \textit{restrict-pool}\ i\ P$
and $i \leq \textit{idx-max}\ (\textit{lq}\ P)$
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
shows $\textit{nz-support}\ (\textit{lq}\ P') \neq \{\}$
⟨*proof*⟩

lemma *restrict-pool-idx-max*:
assumes *finite-liq P*
and $P' = \textit{restrict-pool}\ i\ P$
and $i \leq \textit{idx-max}\ (\textit{lq}\ P)$
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
shows $\textit{idx-max}\ (\textit{lq}\ P) = \textit{idx-max}\ (\textit{lq}\ P')$
⟨*proof*⟩

lemma *restrict-pool-clmm*:
assumes *clmm-dsc P*
and $P' = \textit{restrict-pool}\ i\ P$
shows *clmm-dsc* P'
⟨*proof*⟩

lemma *restrict-pool-quote-gross-leq*:
assumes *mono* (*grd P*)
and $\textit{sqp} \leq \textit{grd}\ P\ i$
and $P' = \textit{restrict-pool}\ i\ P$
shows $\textit{quote-gross}\ P'\ \textit{sqp} = 0$ ⟨*proof*⟩

lemma *restrict-pool-quote-gross*:
assumes *clmm-dsc P*
and $P' = \textit{restrict-pool}\ j\ P$
and $0 < \textit{sqp}$
and $j \leq \textit{lower-tick}\ P\ \textit{sqp}$
shows $\textit{quote-gross}\ P\ \textit{sqp} - \textit{quote-gross}\ P\ (\textit{grd}\ P\ j) = \textit{quote-gross}\ P'\ \textit{sqp}$
⟨*proof*⟩

lemma *restrict-pool-base-net-eq*:
assumes *clmm-dsc P*
and $\textit{nz-support}\ (\textit{lq}\ P) \neq \{\}$
and $P' = \textit{restrict-pool}\ i\ P$

and $i \leq$ *idx-max* (*lq P*)
and *grd P i* $\leq$ *sqp'*
shows *base-net P' sqp'* = *base-net P sqp'*
⟨*proof*⟩

lemma *restrict-pool-grd-min-le*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) $\neq$ {}
and *P'* = *restrict-pool i P*
and $i \leq$ *idx-max* (*lq P*)
shows $i \leq$ *idx-min* (*lq P'*)
⟨*proof*⟩

definition *slice-pool* **where**
slice-pool P sqp = (*let P'* = *refine P sqp in restrict-pool* (*lower-tick P' sqp*) *P'*)

lemma *slice-poolD*:
assumes *P''* = *refine P sqp*
shows *slice-pool P sqp* = *restrict-pool* (*lower-tick P'' sqp*) *P''*
⟨*proof*⟩

lemma *slice-pool-clmm-dsc*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *slice-pool P sqp*
shows *clmm-dsc P'*
⟨*proof*⟩

lemma *slice-pool-nz-liq*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *slice-pool P sqp*
and *lower-tick P sqp* $\leq$ *idx-max* (*lq P*)
and *nz-support* (*lq P*) $\neq$ {}
shows *nz-support* (*lq P'*) $\neq$ {}
⟨*proof*⟩

lemma *slice-pool-tick-idx-max*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *P'* = *slice-pool P sqp*
and *lower-tick P sqp* $\leq$ *idx-max* (*lq P*)
and *nz-support* (*lq P*) $\neq$ {}
shows *lower-tick P' sqp* $\leq$ *idx-max* (*lq P'*)
⟨*proof*⟩

lemma *slice-pool-nz-liq'*:
assumes *clmm-dsc P*
and *P'* = *slice-pool P sqp*

and *nz-support* (*lq P*) ≠ {}
and *0* < *sqp*
and *sqp* < *grd-max P*
shows *nz-support* (*lq P′*) ≠ {}
⟨*proof*⟩

lemma *slice-pool-idx-min*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *nz-support* (*lq P*) ≠ {}
and *P′* = *slice-pool P sqp*
and *i* = *lower-tick P sqp*
and *i* ≤ *idx-max* (*lq P*)
shows *i* ≤ *idx-min* (*lq P′*)
⟨*proof*⟩

lemma *slice-pool-grd-min*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *nz-support* (*lq P*) ≠ {}
and *P′* = *slice-pool P sqp*
and *sqp* < *grd-max P*
shows *sqp* ≤ *grd-min P′*
⟨*proof*⟩

lemma *slice-pool-grd-max*:
assumes *clmm-dsc P*
and *0* < *sqp*
and *nz-support* (*lq P*) ≠ {}
and *P′* = *slice-pool P sqp*
and *lower-tick P sqp* ≤ *idx-max* (*lq P*)
shows *grd-max P* = *grd-max P′* ⟨*proof*⟩

lemma *slice-pool-grd-max′*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *P′* = *slice-pool P sqp*
and *0* < *sqp*
and *sqp* < *grd-max P*
shows *grd-max P* = *grd-max P′*
⟨*proof*⟩

lemma *slice-pool-cst-fees*:
assumes *clmm-dsc P*
and *P′* = *slice-pool P sqp*
and ⋀*i*. *fee P i* = *phi*
shows ⋀*i*. *fee P′ i* = *phi*
⟨*proof*⟩

lemma *slice-pool-quote-gross-leq*:
assumes *clmm-dsc P*
and *0 < sqp*
and *sqp′ ≤ sqp*
and *P′ = slice-pool P sqp*
shows *quote-gross P′ sqp′ = 0*
⟨*proof*⟩

lemma *slice-pool-quote-gross*:
assumes *clmm-dsc P*
and *0 < sqp*
and *sqp ≤ sqp′*
and *P′ = slice-pool P sqp*
shows *quote-gross P′ sqp′ = quote-gross P sqp′ − quote-gross P sqp*
⟨*proof*⟩

lemma *slice-pool-quote-gross-max-eq*:
assumes *clmm-dsc P*
and *P′ = slice-pool P sqp*
and *nz-support (lq P) ≠ {}*
and *0 < sqp*
and *sqp < grd-max P*
and *i = lower-tick P sqp*
and *grd P i = sqp*
shows *quote-gross P′ (grd-max P′) = quote-gross P (grd-max P) − quote-gross P sqp*
⟨*proof*⟩

lemma *slice-pool-quote-gross-inv*:
assumes *clmm-dsc P*
and *0 < sqp*
and *nz-support (lq P) ≠ {}*
and *sqp < grd-max P*
and *0 < y*
and *P′ = slice-pool P sqp*
shows *quote-gross P′ −‘{y} = quote-gross P −‘{y + quote-gross P sqp}*
⟨*proof*⟩

lemma *slice-pool-quote-reach*:
assumes *clmm-dsc P*
and *nz-support (lq P) ≠ {}*
and *0 < sqp*
and *sqp < grd-max P*
and *0 < y*
and *P′ = slice-pool P sqp*
shows *quote-reach P′ y = quote-reach P (y + quote-gross P sqp)*
⟨*proof*⟩

lemma *slice-pool-base-net-eq*:

assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *P′* = *slice-pool P sqp*
and *0* < *sqp*
and *sqp* < *grd-max P*
and *sqp* ≤ *sqp′*
shows *base-net P′ sqp′* = *base-net P sqp′*
⟨*proof*⟩

lemma *slice-pool-base-net-slice*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *i* = *lower-tick P sqp*
and *P′* = *slice-pool P sqp*
and *sqp* < *grd-max P*
and *grd P i* = *sqp*
and *sqp′* ≤ *sqp*
and *0* < *sqp′*
shows *base-net P′ sqp′* = *base-net P′ sqp*
⟨*proof*⟩

lemma *slice-pool-quote-swap-gt-zero*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *grd P* (*lower-tick P sqp2*) = *sqp2*
and *P′* = *slice-pool P sqp2*
and *sqp1* ≤ *sqp2*
and *0* < *y*
and *0* <*sqp1*
and *y* + *quote-gross P sqp2* ≤ *quote-gross P* (*grd-max P*)
shows *quote-swap P′ sqp1 y* = *quote-swap P sqp2 y*
⟨*proof*⟩

lemma *slice-pool-quote-swap*:
assumes *clmm-dsc P*
and *nz-support* (*lq P*) ≠ {}
and *grd P* (*lower-tick P sqp2*) = *sqp2*
and *P′* = *slice-pool P sqp2*
and *sqp1* ≤ *sqp2*
and *sqp2* < *grd-max P*
and *0* ≤ *y*
and *0* < *sqp1*
and *y* + *quote-gross P sqp2* ≤ *quote-gross P* (*grd-max P*)
shows *quote-swap P′ sqp1 y* = *quote-swap P sqp2 y*
⟨*proof*⟩

6.3 CLMM pool join

The join operation is meant to define a pool P on which swap operations can be viewed as a combination of swap operations on its two arguments. We use the convention that the pool fee is 0 on ranges where there is no liquidity.

definition *pool-fee-join* **where**
pool-fee-join P1 P2 i = fee-union (*lq P1 i*) (*lq P2 i*) (*fee P1 i*) (*fee P2 i*)

lemma *pool-fee-join-com*:
shows *pool-fee-join P1 P2 i = pool-fee-join P2 P1 i*
⟨*proof*⟩

definition *joint-pools* **where**
joint-pools P P1 P2 ⟷ (*grd P*) = (*grd P1*) ∧ (*grd P*) = (*grd P2*) ∧
(∀ *i*. *lq P i = lq P1 i + lq P2 i*) ∧
(∀ *i*. *fee P i = pool-fee-join P1 P2 i*)

definition *pool-join* **where**
pool-join P1 P2 =
(*grd P1*, (λ*i*. *lq P1 i + lq P2 i*), (λ*i*. *pool-fee-join P1 P2 i*))

lemma *joint-poolsI*[*intro*]:
assumes *grd P = grd P1*
and *grd P = grd P2*
and ⋀*i*. *lq P i = lq P1 i + lq P2 i*
and ⋀*i*. *fee P i = pool-fee-join P1 P2 i*
shows *joint-pools P P1 P2* ⟨*proof*⟩

lemma *pool-join-joint*:
assumes *grd P1 = grd P2*
and *P = pool-join P1 P2*
shows *joint-pools P P1 P2* ⟨*proof*⟩

lemma *joint-pools-grids*:
assumes *joint-pools P P1 P2*
shows (*grd P*) = (*grd P1*) (*grd P*) = (*grd P2*)
⟨*proof*⟩

lemma *joint-pools-lq*:
assumes *joint-pools P P1 P2*
shows *lq P i = lq P1 i + lq P2 i*
⟨*proof*⟩

lemma *joint-pools-fee*:
assumes *joint-pools P P1 P2*
shows *fee P i = pool-fee-join P1 P2 i*
⟨*proof*⟩

lemma *joint-pools-com*:
assumes *joint-pools P P1 P2*
shows *joint-pools P P2 P1*
⟨*proof*⟩

lemma *joint-pools-nz-liq-sub*:
assumes *joint-pools P P1 P2*
shows *nz-support* (*lq P*) ⊆ *nz-support* (*lq P1*) ∪ (*nz-support* (*lq P2*))
⟨*proof*⟩

lemma *joint-pools-nz-liq-sup*:
assumes *joint-pools P P1 P2*
and ⋀*i*. *0* ≤ *lq P1 i*
and ⋀*i*. *0* ≤ *lq P2 i*
shows *nz-support* (*lq P1*) ∪ (*nz-support* (*lq P2*)) ⊆ *nz-support* (*lq P*)
⟨*proof*⟩

lemma *joint-pools-nz-liq*:
assumes *joint-pools P P1 P2*
and ⋀*i*. *0* ≤ *lq P1 i*
and ⋀*i*. *0* ≤ *lq P2 i*
shows *nz-support* (*lq P1*) ∪ (*nz-support* (*lq P2*)) = *nz-support* (*lq P*)
⟨*proof*⟩

lemma *clmm-joint-pools-nz-liq*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *nz-support* (*lq P1*) ∪ (*nz-support* (*lq P2*)) = *nz-support* (*lq P*)
⟨*proof*⟩

lemma *joint-pools-finite-liq*:
assumes *finite-liq P1*
and *finite-liq P2*
and *joint-pools P P1 P2*
shows *finite-liq P* ⟨*proof*⟩

lemma *joint-pools-idx-min-min*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P1*) ≠ {}
and *idx-min* (*lq P1*) ≤ *idx-min* (*lq P2*)
shows *idx-min* (*lq P*) = *idx-min* (*lq P1*)
⟨*proof*⟩

lemma *joint-pools-idx-min*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*

and *joint-pools P P1 P2*
and *nz-support* (*lq P1*) ≠ {}
and *nz-support* (*lq P2*) ≠ {}
shows *idx-min* (*lq P*) = *min* (*idx-min* (*lq P1*)) (*idx-min* (*lq P2*))
⟨*proof*⟩

lemma *joint-pools-idx-max-max*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P2*) ≠ {}
and *idx-max* (*lq P1*) ≤ *idx-max* (*lq P2*)
shows *idx-max* (*lq P*) = *idx-max* (*lq P2*)
⟨*proof*⟩

lemma *joint-pools-idx-max*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P1*) ≠ {}
and *nz-support* (*lq P2*) ≠ {}
shows *idx-max* (*lq P*) = *max* (*idx-max* (*lq P1*)) (*idx-max* (*lq P2*))
⟨*proof*⟩

lemma *joint-pools-clmm-dsc*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *clmm-dsc P*
⟨*proof*⟩

lemma *join-gross-fct*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *gross-fct* (*lq P*) (*fee P*) *i* = *gross-fct* (*lq P1*) (*fee P1*) *i* +
gross-fct (*lq P2*) (*fee P2*) *i*
⟨*proof*⟩

lemma *quote-gross-join*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *quote-gross P x* = *quote-gross P1 x* + *quote-gross P2 x*
⟨*proof*⟩

lemma *quote-net-join*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*

and *joint-pools P P1 P2*
shows *quote-net P x = quote-net P1 x + quote-net P2 x*
⟨*proof*⟩

lemma *base-gross-join*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *base-gross P x = base-gross P1 x + base-gross P2 x*
⟨*proof*⟩

lemma *base-net-join*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *base-net P x = base-net P1 x + base-net P2 x*
⟨*proof*⟩

lemma *mkt-depth-join*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
shows *mkt-depth P x x′ = mkt-depth P1 x x′ + mkt-depth P2 x x′*
⟨*proof*⟩

lemma *joint-quote-gross-decomp*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P*) ≠ {}
and *grd-min P* ≤ *x*
and *0* ≤ *y*
and *y + quote-gross P x* ≤ *quote-gross P* (*grd-max P*)
and *x′ = quote-reach P* (*y + quote-gross P x*)
and *y1 = quote-gross P1 x′ − quote-gross P1 x*
and *y2 = quote-gross P2 x′ − quote-gross P2 x*
shows *y = y1 + y2*
⟨*proof*⟩

lemma *joint-quote-gross-decomp′*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P*) ≠ {}
and *0* ≤ *y*
and *y* ≤ *quote-gross P* (*grd-max P*)
and *x′ = quote-reach P y*
and *y1 = quote-gross P1 x′*
and *y2 = quote-gross P2 x′*

shows $y = y1 + y2$
⟨*proof*⟩

lemma *joint-base-net-decomp'*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P*) ≠ {}
and $0 \leq y$
and $y \leq$ *quote-gross P* (*grd-max P*)
and $x' =$ *quote-reach P y*
and *y1* = *base-net P1 x'*
and *y2* = *base-net P2 x'*
shows *base-net P x'* = *y1* + *y2*
⟨*proof*⟩

lemma *joint-base-gross-decomp*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *joint-pools P P1 P2*
and *nz-support* (*lq P*) ≠ {}
and $x \leq$ *grd-max P*
and $0 \leq y$
and $y +$ *base-gross P x* ≤ *base-gross P* (*grd-min P*)
and $x' =$ *base-reach P* ($y +$ *base-gross P x*)
and *y1* = *base-gross P1 x'* − *base-gross P1 x*
and *y2* = *base-gross P2 x'* − *base-gross P2 x*
shows *y* = *y1* + *y2*
⟨*proof*⟩

definition *join-pools* **where**
join-pools P1 P2 =
(*grd P1*,
(λ*i*. *lq P1 i* + *lq P2 i*),
(λ*i*. *pool-fee-join P1 P2 i*))

lemma *join-pools-grd*[*simp*]:
assumes *P* = *join-pools P1 P2*
shows *grd P* = *grd P1* ⟨*proof*⟩

lemma *join-pools-lq*[*simp*]:
assumes *P* = *join-pools P1 P2*
shows *lq P i* = *lq P1 i* + *lq P2 i*
⟨*proof*⟩

lemma *join-pools-fee*[*simp*]:
assumes *P* = *join-pools P1 P2*
shows *fee P i* = *pool-fee-join P1 P2 i*
⟨*proof*⟩

lemma *join-joint-pools*:
assumes *grd P1 = grd P2*
shows *joint-pools* (*join-pools P1 P2*) *P1 P2*
⟨*proof*⟩

6.4 CLMM pool combination

definition *pool-comb* **where**
pool-comb P1 P2 sqp = (*let P′* = *refine P1 sqp in*
pool-join P′ (*slice-pool P2 sqp*))

lemma *pool-comb-joint*:
assumes *grd P1 = grd P2*
shows *joint-pools* (*pool-comb P1 P2 sqp*) (*refine P1 sqp*)
(*slice-pool P2 sqp*) ⟨*proof*⟩

lemma *pool-comb-refined-joint-nz-liq*:
assumes *grd P1 = grd P2*
and *clmm-dsc P1*
and *clmm-dsc P2*
and *P = pool-comb P1 P2 sqp*
and *grd P1* (*lower-tick P1 sqp*) = *sqp*
shows *nz-support* (*lq P*) = *nz-support* (*lq P1*) ∪
(*nz-support* (*lq* (*slice-pool P2 sqp*)))
⟨*proof*⟩

lemma *pool-comb-joint-refined*:
assumes *grd P1 = grd P2*
and *grd P1* (*lower-tick P1 sqp*) = *sqp*
shows *joint-pools* (*pool-comb P1 P2 sqp*) *P1*
(*slice-pool P2 sqp*)
⟨*proof*⟩

lemma *pool-comb-clmm-dsc*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1 = grd P2*
and *0 < sqp*
and *P3 = pool-comb P1 P2 sqp*
shows *clmm-dsc P3* ⟨*proof*⟩

lemma *pool-comb-grd-min*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1 = grd P2*
and *nz-support* (*lq P1*) ≠ {}
and *nz-support* (*lq P2*) ≠ {}
and *0 < sqp*

and *sqp* < *grd-max P2*
and *P* = *pool-comb P1 P2 sqp*
shows *grd-min P* = *min* (*grd-min P1*) (*grd-min* (*slice-pool P2 sqp*))
⟨*proof*⟩

lemma *pool-comb-le-grd-min*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1* = *grd P2*
and *nz-support* (*lq P1*) ≠ {}
and *nz-support* (*lq P2*) ≠ {}
and *0* < *sqp*
and *sqp* < *grd-max P2*
and *grd-min P1* ≤ *sqp*
and *P* = *pool-comb P1 P2 sqp*
shows *grd-min P* = *grd-min P1*
⟨*proof*⟩

lemma *pool-comb-grd-max*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1* = *grd P2*
and *nz-support* (*lq P1*) ≠ {}
and *nz-support* (*lq P2*) ≠ {}
and *0* < *sqp*
and *sqp* < *grd-max P2*
and *P* = *pool-comb P1 P2 sqp*
shows *grd-max P* = *max* (*grd-max P1*) (*grd-max P2*)
⟨*proof*⟩

lemma *pool-comb-grd-max-slice*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1* = *grd P2*
and *nz-support* (*lq P1*) ≠ {}
and *nz-support* (*lq P2*) ≠ {}
and *0* < *sqp*
and *sqp* < *grd-max P2*
and *P* = *pool-comb P1 P2 sqp*
shows *sqp* < *grd-max P*
⟨*proof*⟩

lemma *pool-comb-quote-decomp*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1* = *grd P2*
and *grd P1* (*lower-tick P1 sqp*) = *sqp*
and *0* < *sqp*
and *sqp′* ≤ *grd-max P*

and $P = \textit{pool-comb P1 P2 sqp}$
and $\textit{nz-support } (\textit{lq } P) \neq \{\}$
shows $\textit{quote-gross } P \textit{ sqp}' = \textit{quote-gross P1 sqp}' + \textit{quote-gross } (\textit{slice-pool P2 sqp}) \textit{ sqp}'$
⟨*proof*⟩

lemma *pool-comb-quote-le-slice*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and $\textit{grd P1} = \textit{grd P2}$
and $\textit{grd P1 } (\textit{lower-tick P1 sqp}) = \textit{sqp}$
and $0 < \textit{sqp}$
and $\textit{sqp}' \leq \textit{sqp}$
and $\textit{sqp} \leq \textit{grd-max } P$
and $P = \textit{pool-comb P1 P2 sqp}$
and $\textit{nz-support } (\textit{lq } P) \neq \{\}$
shows $\textit{quote-gross } P \textit{ sqp}' = \textit{quote-gross P1 sqp}'$
⟨*proof*⟩

lemma *pool-comb-quote-diff-decomp*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and $\textit{grd P1} = \textit{grd P2}$
and $\textit{grd P1 } (\textit{lower-tick P1 sqp}) = \textit{sqp}$
and $0 < \textit{sqp}$
and $0 < \textit{sqp}'$
and $0 < \textit{sqp1}$
and $\textit{sqp}' \leq \textit{grd-max } P$
and $\textit{sqp1} \leq \textit{grd-max } P$
and $P = \textit{pool-comb P1 P2 sqp}$
and $\textit{nz-support } (\textit{lq } P) \neq \{\}$
shows $\textit{quote-gross } P \textit{ sqp}' - \textit{quote-gross } P \textit{ sqp1} =$
$\textit{quote-gross P1 sqp}' - \textit{quote-gross P1 sqp1} +$
$\textit{quote-gross } (\textit{slice-pool P2 sqp}) \textit{ sqp}' - \textit{quote-gross } (\textit{slice-pool P2 sqp}) \textit{ sqp1}$
⟨*proof*⟩

lemma *pool-comb-base-net-plus*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and $\textit{grd P1} = \textit{grd P2}$
and $\textit{grd P1 } (\textit{lower-tick P1 sqp2}) = \textit{sqp2}$
and $0 < \textit{sqp2}$
and $0 < y$
and $y \leq \textit{quote-gross } P \textit{ (grd-max } P)$
and $P = \textit{pool-comb P1 P2 sqp2}$
and $\textit{sqp}' = \textit{quote-reach } P \textit{ } y$
and $\textit{sqp}' \leq \textit{sqp2}$
and $\textit{nz-support } (\textit{lq } P) \neq \{\}$
shows $\textit{base-net } P \textit{ sqp}' = \textit{base-net P1 sqp}' + \textit{base-net } (\textit{slice-pool P2 sqp2}) \textit{ sqp}'$

⟨*proof*⟩

lemma *combo-quote-init1*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1 = grd P2*
and *grd P1 (lower-tick P1 sqp2) = sqp2*
and *0 < sqp2*
and *P = pool-comb P1 P2 sqp2*
and *0 < y*
and *nz-support (lq P1) ≠ {}*
and *nz-support (lq P2) ≠ {}*
and *y + quote-gross P sqp1 ≤ quote-gross P (grd-max P)*
and *sqp′ = quote-reach P (y + quote-gross P sqp1)*
and *sqp2 < grd-max P2*
and *sqp1 ≤ sqp2*
shows *quote-gross P sqp1 = quote-gross P1 sqp1*
⟨*proof*⟩

lemma *combo-remain-quote-eq*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1 = grd P2*
and *grd P1 (lower-tick P1 sqp2) = sqp2*
and *0 < sqp2*
and *P = pool-comb P1 P2 sqp2*
and *nz-support (lq P) ≠ {}*
and *nz-support (lq P2) ≠ {}*
and *0 < y*
and *0< sqp1*
and *y + quote-gross P sqp1 ≤ quote-gross P (grd-max P)*
and *sqp′ = quote-reach P (y + quote-gross P sqp1)*
and *y1 = quote-gross P1 sqp′ − quote-gross P1 sqp1*
and *sqp2 ≤ sqp′*
and *sqp1 ≤ sqp2*
and *rs1′ = quote-reach P2 (y − y1 + quote-gross P2 sqp2)*
shows *quote-gross P2 sqp′ = quote-gross P2 rs1′*
⟨*proof*⟩

lemma *comb-quote-gross-le*:
assumes *clmm-dsc P1*
and *clmm-dsc P2*
and *grd P1 = grd P2*
and *0 < sqp*
and *sqp < grd-max P*
and *0 < y*
and *y ≤ quote-gross P sqp*
and *y ≤ quote-gross P (grd-max P)*
and *P = pool-comb P1 P2 sqp*

and $sqp' = quote\text{-}reach\ P\ y$
and $nz\text{-}support\ (lq\ P) \neq \{\}$
shows $quote\text{-}gross\ P1\ sqp' = y$
⟨*proof*⟩

locale *combined-pools* =
fixes *P1 P2 P sqp2*
assumes *cmb-P1*: *clmm-dsc P1*
and *cmb-P2*: *clmm-dsc P2*
and *cmb-grd-eq*: *grd P1 = grd P2*
and *cmb-on-grid*: $grd\ P1\ (lower\text{-}tick\ P1\ sqp2) = sqp2$
and *cmb-nz1*: $nz\text{-}support\ (lq\ P1) \neq \{\}$
and *cmb-nz2*: $nz\text{-}support\ (lq\ P2) \neq \{\}$
and *cmb-comb*: *P = pool-comb P1 P2 sqp2*
and *cmb-pos*: *0 < sqp2*
and *cmb-max*: *sqp2 < grd-max P2*

begin

lemma *combined-P-prop*:
shows $clmm\text{-}dsc\ P\ nz\text{-}support\ (lq\ P) \neq \{\}$
⟨*proof*⟩

lemmas *cmb-props = cmb-P1 cmb-P2 cmb-grd-eq cmb-on-grid cmb-nz1 cmb-nz2 cmb-comb cmb-pos cmb-max combined-P-prop*

lemma *combo-joint-quote-gross-decomp*:
assumes *0 < y*
and *0 < sqp1*
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $P'' = slice\text{-}pool\ P2\ sqp2$
and $y2' = quote\text{-}gross\ P''\ sqp' - quote\text{-}gross\ P''\ sqp1$
shows $y = y1 + y2'\ y1 \leq y\ 0 \leq y1$
$y1 + quote\text{-}gross\ P1\ sqp1 \leq quote\text{-}gross\ P1\ (grd\text{-}max\ P1)$
$y2' \leq quote\text{-}gross\ P''\ (grd\text{-}max\ P2)$
⟨*proof*⟩

lemma *combo-joint-quote-gross-leq-max*:
assumes *0 < y*
and *0 < sqp1*
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
shows $y - y1 + quote\text{-}gross\ P2\ sqp2 \leq quote\text{-}gross\ P2\ (grd\text{-}max\ P2)$
⟨*proof*⟩

lemma *combo-joint-quote-gross-price-le*:

assumes $0 < y$
and *grd-min P1* $\leq$ *sqp1*
and *sqp1* $\leq$ *sqp2*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp'* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp'* − *quote-gross P1 sqp1*
and *rs1* = *quote-reach P1* (*y1* + *quote-gross P1 sqp1*)
shows *rs1* $\leq$ *sqp'*
⟨*proof*⟩

lemma *combo-joint-quote-gross-decomp-leq*:
assumes $0 < y$
and $0 <$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp'* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp'* − *quote-gross P1 sqp1*
and *P''* = *slice-pool P2 sqp2*
and *sqp1* $\leq$ *sqp2*
and *y2'* = *quote-gross P''* *sqp'*
shows y = *y1* + *y2'* *y1* $\leq$ y $0 \leq$ *y1*
y1 + *quote-gross P1 sqp1* $\leq$ *quote-gross P1* (*grd-max P1*)
y2' $\leq$ *quote-gross P''* (*grd-max P2*)
⟨*proof*⟩

lemma *combo-quote-swap-slice-eq*:
assumes $0 <$ *sqp1*
and $0 < y$
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp'* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp'* − *quote-gross P1 sqp1*
shows *quote-swap P sqp1 y* = *quote-swap P1 sqp1 y1* +
quote-swap (*slice-pool P2 sqp2*) *sqp1* (y − *y1*)
⟨*proof*⟩

lemma *combo-quote-swap-eq*:
assumes $0 <$ *sqp1*
and $0 < y$
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp'* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp'* − *quote-gross P1 sqp1*
and *sqp1* $\leq$ *sqp2*
shows *quote-swap P sqp1 y* = *quote-swap P1 sqp1 y1* +
quote-swap P2 sqp2 (y − *y1*)
⟨*proof*⟩

lemma *comb-add-above-gt*:
assumes $0 < y$
and $0 <$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)

and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $y1 < y$
and $sqp1 \leq sqp2$
shows $sqp2 < sqp'$
⟨*proof*⟩

lemma *comb-add-above-add-eq*:
assumes $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $rs1 = quote\text{-}reach\ P1\ (y1 + quote\text{-}gross\ P1\ sqp1)$
shows $quote\text{-}gross\ P1\ sqp' = quote\text{-}gross\ P1\ rs1$
⟨*proof*⟩

lemma *comb-add-above-add-eq2*:
assumes $0 < y$
and $grd\text{-}min\ P1 \leq sqp1$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $sqp1 \leq sqp2$
and $y1 < y$
and $rs1' = quote\text{-}reach\ P2\ (y - y1 + quote\text{-}gross\ P2\ sqp2)$
shows $quote\text{-}gross\ P2\ sqp' = quote\text{-}gross\ P2\ rs1'$
⟨*proof*⟩

lemma *combo-joint-rest-qty-slice*:
assumes $0 < y$
and $0 < sqp1$
and $sqp1 \leq sqp2$
and $y + quote\text{-}gross\ P\ sqp1 < quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $P'' = slice\text{-}pool\ P2\ sqp2$
shows $y - y1 = quote\text{-}gross\ P''\ sqp'$
⟨*proof*⟩

lemma *combo-joint-rest-qty*:
assumes $0 < y$
and $0 < sqp1$
and $sqp1 \leq sqp2$
and $y + quote\text{-}gross\ P\ sqp1 < quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $sqp2 \leq sqp'$
shows $y - y1 = quote\text{-}gross\ P2\ sqp' - quote\text{-}gross\ P2\ sqp2$
⟨*proof*⟩

lemma *combo-joint-rest-qty-le*:
assumes $0 < y$

and $0 < sqp1$
and $sqp1 \leq sqp2$
and $y + quote\text{-}gross\ P\ sqp1 < quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
shows $y - y1 + quote\text{-}gross\ P2\ sqp2 \leq quote\text{-}gross\ P2\ (grd\text{-}max\ P2)$
⟨*proof*⟩

lemma *combo-joint-rest-price-pos*:
assumes $0 < y$
and $grd\text{-}min\ P1 \leq sqp1$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $rs1' = quote\text{-}reach\ P2\ (y - y1 + quote\text{-}gross\ P2\ sqp2)$
and $y1 < y$
shows $0 < rs1'$
⟨*proof*⟩

lemma *combo-joint-quote-gross-price-le'*:
assumes $0 < y$
and $grd\text{-}min\ P1 \leq sqp1$
and $sqp1 \leq sqp2$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $y1 < y$
and $rs1' = quote\text{-}reach\ P2\ (y - y1 + quote\text{-}gross\ P2\ sqp2)$
shows $rs1' \leq sqp'$
⟨*proof*⟩

lemma *comb-add-above-price1-leq*:
assumes $0 < y$
and $grd\text{-}min\ P1 \leq sqp1$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $sqp1 \leq sqp2$
and $0 \leq y2$
and $y - y2 + quote\text{-}gross\ P2\ sqp2 \leq quote\text{-}gross\ P2\ (grd\text{-}max\ P2)$
and $y2 < y1$
and $y1 < y$
and $rs1 = quote\text{-}reach\ P1\ (y1 + quote\text{-}gross\ P1\ sqp1)$
and $rs2 = quote\text{-}reach\ P1\ (y2 + quote\text{-}gross\ P1\ sqp1)$
shows $rs2 \leq rs1$
⟨*proof*⟩

lemma *comb-add-above-price2-geq*:
assumes $0 < y$

and *grd-min P1* ≤ *sqp1*
and *y* + *quote-gross P sqp1* ≤ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (*y* + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* ≤ *sqp2*
and *0* ≤ *y2*
and *y* − *y2* + *quote-gross P2 sqp2* ≤ *quote-gross P2* (*grd-max P2*)
and *y2* < *y1*
and *y1* < *y*
and *rs1′* = *quote-reach P2* (*y* − *y1* + *quote-gross P2 sqp2*)
and *rs2′* = *quote-reach P2* (*y* − *y2* + *quote-gross P2 sqp2*)
shows *rs1′* ≤ *rs2′*
⟨*proof*⟩

lemma *comb-add-above-price2-geq′*:
assumes *0* < *y*
and *grd-min P1* ≤ *sqp1*
and *y* + *quote-gross P sqp1* ≤ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (*y* + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* ≤ *sqp2*
and *0* ≤ *y2*
and *y* − *y1* + *quote-gross P2 sqp2* ≤ *quote-gross P2* (*grd-max P2*)
and *y1* < *y2*
and *y2* ≤ *y*
and *rs1′* = *quote-reach P2* (*y* − *y1* + *quote-gross P2 sqp2*)
and *rs2′* = *quote-reach P2* (*y* − *y2* + *quote-gross P2 sqp2*)
shows *rs2′* ≤ *rs1′*
⟨*proof*⟩

lemma *comb-add-above-price2-lt*:
assumes *0* < *y*
and *grd-min P1* ≤ *sqp1*
and *y* + *quote-gross P sqp1* ≤ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (*y* + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* ≤ *sqp2*
and *0* ≤ *y2*
and *y* − *y2* + *quote-gross P2 sqp2* ≤ *quote-gross P2* (*grd-max P2*)
and *y2* < *y1*
and *y1* < *y*
and *rs2′* = *quote-reach P2* (*y* − *y2* + *quote-gross P2 sqp2*)
shows *sqp′* < *rs2′*
⟨*proof*⟩

lemma *combo-joint-reached-price-pos*:
assumes *0* < *y*
and *y* + *quote-gross P sqp1* ≤ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (*y* + *quote-gross P sqp1*)

shows $0 < sqp'$ ⟨*proof*⟩

lemma *combo-joint-base-reached-eq*:
assumes $0 < y$
and *grd-min P1* $\leq$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and sqp' = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* $\leq$ *sqp2*
and *rs1* = *quote-reach P1* (*y1* + *quote-gross P1 sqp1*)
shows *base-net P1 sqp′* = *base-net P1 rs1*
⟨*proof*⟩

lemma *combo-joint-base-reached-eq2*:
assumes $0 < y$
and *grd-min P1* $\leq$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and sqp' = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* $\leq$ *sqp2*
and *y1* < y
and *rs1′* = *quote-reach P2* (y − *y1* + *quote-gross P2 sqp2*)
shows *base-net P2 sqp′* = *base-net P2 rs1′*
⟨*proof*⟩

lemma *quote-gross-price-eq1*:
assumes *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *rs1* = *quote-reach P1* (*y1* + *quote-gross P1 sqp1*)
shows *quote-gross P1 rs1* = *y1* + *quote-gross P1 sqp1*
⟨*proof*⟩

lemma *quote-gross-price-eq2*:
assumes $0 \leq$ *y2*
and *y2* + *quote-gross P1 sqp1* $\leq$ *quote-gross P1* (*grd-max P1*)
and *rs2* = *quote-reach P1* (*y2* + *quote-gross P1 sqp1*)
shows *quote-gross P1 rs2* = *y2* + *quote-gross P1 sqp1*
⟨*proof*⟩

end

6.5 Optimality result on quote tokens

When the fees in two pools are constant and equal, swapping a given amount of quote tokens in their combination permits to determine the optimal quantities of quote tokens to swap in each individual pool.

locale *combined-pools-cst-fee* = *combined-pools* +
fixes *phi*
assumes *fee1*: $\forall i$. *fee P1 i* = *phi*
and *fee2*: $\forall i$. *fee P2 i* = *phi*

begin

lemma *fee-props*:
shows $0 \leq phi\ phi < 1$ ⟨*proof*⟩

lemma *quote-swap-opt-above*:
assumes $0 < y$
and $grd\text{-}min\ P1 \leq sqp1$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $sqp1 \leq sqp2$
and $0 \leq y2$
and $y - y2 + quote\text{-}gross\ P2\ sqp2 \leq quote\text{-}gross\ P2\ (grd\text{-}max\ P2)$
and $y2 < y1$
and $y1 < y$
shows $quote\text{-}swap\ P1\ sqp1\ y2 + quote\text{-}swap\ P2\ sqp2\ (y - y2) \leq quote\text{-}swap\ P\ sqp1\ y$
⟨*proof*⟩

lemma *quote-swap-opt-above'*:
assumes $0 < y$
and $grd\text{-}min\ P1 \leq sqp1$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $sqp1 \leq sqp2$
and $y2 + quote\text{-}gross\ P1\ sqp1 \leq quote\text{-}gross\ P1\ (grd\text{-}max\ P1)$
and $0 \leq y - y2$
and $y1 < y2$
and $y2 \leq y$
shows $quote\text{-}swap\ P1\ sqp1\ y2 + quote\text{-}swap\ P2\ sqp2\ (y - y2) \leq quote\text{-}swap\ P\ sqp1\ y$
⟨*proof*⟩

lemma *combo-slice-no-addition2*:
assumes $0 < y$
and $y + quote\text{-}gross\ P\ sqp1 \leq quote\text{-}gross\ P\ (grd\text{-}max\ P)$
and $sqp' = quote\text{-}reach\ P\ (y + quote\text{-}gross\ P\ sqp1)$
and $y1 = quote\text{-}gross\ P1\ sqp' - quote\text{-}gross\ P1\ sqp1$
and $sqp1 \leq sqp2$
and $y1 = y$
and $0 \leq y2$
and $y2 \leq y$
and $y - y2 + quote\text{-}gross\ P2\ sqp2 \leq quote\text{-}gross\ P2\ (grd\text{-}max\ P2)$
and $y1 \neq y2$
and $P'' = slice\text{-}pool\ P2\ sqp2$
shows $quote\text{-}gross\ P''\ sqp' = 0$

⟨*proof*⟩

lemma *quote-swap-opt-below*:
assumes $0 < y$
and *grd-min P1* $\leq$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* $\leq$ *sqp2*
and *y1* = y
and $0 \leq$ *y2*
and *y2* $\leq y$
and y − *y2* + *quote-gross P2 sqp2* $\leq$ *quote-gross P2* (*grd-max P2*)
and *y1* $\neq$ *y2*
shows *quote-swap P1 sqp1 y2* + *quote-swap P2 sqp2* (y − *y2*) $\leq$ *quote-swap P sqp1 y*
⟨*proof*⟩

lemma *quote-swap-optimum′*:
assumes $0 < y$
and *grd-min P1* $\leq$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* $\leq$ *sqp2*
and $0 \leq$ *y2*
and *y2* $\leq y$
and *y2* + *quote-gross P1 sqp1* $\leq$ *quote-gross P1* (*grd-max P1*)
and y − *y2* + *quote-gross P2 sqp2* $\leq$ *quote-gross P2* (*grd-max P2*)
and *y1* $\neq$ *y2*
shows *quote-swap P1 sqp1 y2* + *quote-swap P2 sqp2* (y − *y2*) $\leq$ *quote-swap P sqp1 y*
⟨*proof*⟩

lemma *quote-swap-optimum*:
assumes $0 < y$
and $0 <$ *sqp1*
and y + *quote-gross P sqp1* $\leq$ *quote-gross P* (*grd-max P*)
and *sqp′* = *quote-reach P* (y + *quote-gross P sqp1*)
and *y1* = *quote-gross P1 sqp′* − *quote-gross P1 sqp1*
and *sqp1* $\leq$ *sqp2*
and *grd-min P1* $\leq$ *sqp2*
and $0 \leq$ *y2*
and *y2* $\leq y$
and *y2* + *quote-gross P1 sqp1* $\leq$ *quote-gross P1* (*grd-max P1*)
and y − *y2* + *quote-gross P2 sqp2* $\leq$ *quote-gross P2* (*grd-max P2*)
and *y1* $\neq$ *y2*
shows *quote-swap P1 sqp1 y2* + *quote-swap P2 sqp2* (y − *y2*) $\leq$ *quote-swap P sqp1 y*

⟨*proof*⟩

end

end

References

[1] M. Echenim, E. Gobet, and A.-C. Maurice. Uniswap v3: impermanent loss modeling and swap fees asymptotic analysis, Sept. 2025. Available at https://hal.science/hal-04214315v3/file/Article_IL_Uniswapv3_revision_HAL.pdf.