Settlement Algebra and Verified Constant-Product Invariants for Batch DEX Execution

Dana Edwards

January 30, 2026

Abstract

We present a small, compositional mathematical model for decentralized exchange (DEX) execution and prove key safety invariants for constant-product market makers (CPMMs), including a closed-form expression for the *exact* per-swap increase in the product invariant. The development is mechanized in Lean 4/Mathlib.

The paper has three main layers: (i) an additive *settlement algebra* capturing token flows as a commutative group, with a homomorphism Δ to a scalar "net flow"; (ii) a correctness-by-construction objective for batch auctions maximizing executed volume and surplus in lexicographic order; and (iii) a CPMM state-transition model with verified K-monotonicity and an exact "K-gap" remainder theorem, lifted compositionally to batches and bridged to the settlement/objective layers by a refinement lemma.

1 Overview and scope

This draft explains the mathematics and formal logic embodied by the Lean files:

- `lean-mathlib/Proofs/SettlementAlgebra.lean`: additive settlement model and the homomorphism Δ.
- `lean-mathlib/Proofs/BatchOptimality.lean`: batch objective (A, B) (volume, surplus) and lexicographic comparison.
- `lean-mathlib/Proofs/CPMMInvariants.lean`: CPMM safety theorems, including the closed-form K-gap.
- `lean-mathlib/Proofs/CPMMSettlement.lean`: bridge from CPMM swaps to settlements.
- `lean-mathlib/Proofs/BatchCPMMUnification.lean`: unification: batch settlement safety, batch optimality additivity, and compositional K theorems for sequential swap batches.

Important semantic caveat. Throughout, we distinguish:

- *Scalar net-flow* $\Delta(s) = \Delta x + \Delta y$ which adds different token units and is therefore *not* a notion of economic value conservation; and
- the *CPMM invariant* $K = x \cdot y$ which is the standard, economically meaningful conservation law for constant-product pools.

The Lean development makes this distinction explicit and proves K-monotonicity as the primary invariant.

2 Preliminaries

We write $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{Z}$ for the integers. For $d \in \mathbb{N}$ with $d > 0$, Euclidean division states that for every $N \in \mathbb{N}$ there exist unique $q, r \in \mathbb{N}$ such that

$$N = dq + r \quad \text{and} \quad 0 \le r < d, \tag{1}$$

where $q = \lfloor N/d \rfloor$ and $r = N \bmod d$ [1].

3 Settlement algebra

3.1 Definition

Definition 1 (Settlement). *Define the* settlement space *as*

$$\mathsf{Settlement} := \mathbb{Z} \times \mathbb{Z}.$$

An element $s = (\Delta x, \Delta y)$ represents signed changes to two pool reserves. Positive means inflow to the pool*, negative means* outflow*.*

Definition 2 (Composition and identity). *Define composition as componentwise addition:*

$$(\Delta x_1, \Delta y_1) \circ_s (\Delta x_2, \Delta y_2) := (\Delta x_1 + \Delta x_2,\ \Delta y_1 + \Delta y_2).$$

The identity settlement is $0 := (0, 0)$.

Proposition 1 (Abelian group structure). *$(\mathsf{Settlement}, \circ_s, 0)$ is an abelian group (indeed an* $\mathsf{AddCommGroup}$ *in Mathlib).*

Proof. Componentwise integer addition is associative and commutative, with identity $(0, 0)$ and inverse $-(\Delta x, \Delta y) = (-\Delta x, -\Delta y)$. □

3.2 Net-flow homomorphism and balance

Definition 3 (Net scalar flow). *Define $\Delta : \mathsf{Settlement} \to \mathbb{Z}$ by*

$$\Delta(\Delta x, \Delta y) := \Delta x + \Delta y.$$

Proposition 2 (Homomorphism). *Δ is a group homomorphism:*

$$\Delta(s_1 \circ_s s_2) = \Delta(s_1) + \Delta(s_2), \qquad \Delta(0) = 0, \qquad \Delta(-s) = -\Delta(s).$$

Proof. Immediate from distributivity of addition in $\mathbb{Z}$:

$$\Delta((\Delta x_1 + \Delta x_2, \Delta y_1 + \Delta y_2)) = (\Delta x_1 + \Delta x_2) + (\Delta y_1 + \Delta y_2) = (\Delta x_1 + \Delta y_1) + (\Delta x_2 + \Delta y_2).$$

□

Definition 4 (Balanced settlements). *Call a settlement* balanced *if $\Delta(s) = 0$. Equivalently,*

$$\mathsf{Balanced} := \ker(\Delta) \subseteq \mathsf{Settlement}.$$

Proposition 3 (Kernel is a subgroup). $\mathsf{Balanced}$ *is an additive subgroup of* $\mathsf{Settlement}$*.*

Proof. Kernels of homomorphisms are subgroups; explicitly: if $\Delta(s_1) = 0$ and $\Delta(s_2) = 0$, then $\Delta(s_1 \circ_s s_2) = 0$, and if $\Delta(s) = 0$ then $\Delta(-s) = 0$. $\square$

Remark 1 ("Safety" as a scalar predicate). The Lean development also defines a *scalar safety* predicate

$$\mathsf{isSafe}(s) \iff \Delta(s) \geq 0.$$

This is closed under composition by the homomorphism property and monotonicity of $+$ on $\mathbb{Z}$, but it should not be conflated with economic safety, since it adds different token units.

3.3 Swap-to-settlement embedding

Definition 5 (Swap settlement)**.** *Given natural amounts* $\mathsf{amt}_{\mathrm{in}}, \mathsf{amt}_{\mathrm{out}} \in \mathbb{N}$*, define the swap settlement (pool perspective)*

$$\mathsf{swap}(\mathsf{amt}_{\mathrm{in}}, \mathsf{amt}_{\mathrm{out}}) \coloneqq (\mathsf{amt}_{\mathrm{in}},\ -\mathsf{amt}_{\mathrm{out}}) \in \mathbb{Z} \times \mathbb{Z}.$$

Then

$$\Delta(\mathsf{swap}(\mathsf{amt}_{\mathrm{in}}, \mathsf{amt}_{\mathrm{out}})) = \mathsf{amt}_{\mathrm{in}} - \mathsf{amt}_{\mathrm{out}}.$$

In particular, if $\mathsf{amt}_{\mathrm{in}} = \mathsf{amt}_{\mathrm{out}}$ then the settlement is balanced.

4 Batch optimality: maximizing volume and surplus

This section formalizes an objective for batch execution that is *correctness-by-construction*: invalid fills (output below minimum) are unrepresentable.

4.1 Intents and valid outcomes

Definition 6 (Intent)**.** *An* intent *is a pair* $i = (a, m) \in \mathbb{N} \times \mathbb{N}$*, where* a *is the input amount and* m *is the minimum acceptable output.*

Definition 7 (Valid outcome)**.** *For a fixed intent* $i = (a, m)$*, a* valid outcome *is either:*

1. *unfilled, or*
2. *filled*(o) *for some* $o \in \mathbb{N}$ *with the proof obligation* $o \geq m$*.*

This is implemented as a dependent inductive type in Lean (`ValidOutcome i`).

4.2 Objective functions

Definition 8 (Volume and surplus)**.** *For intent* $i = (a, m)$ *and valid outcome* o*:*

$$\mathsf{volume}(o) \coloneqq \begin{cases} 0 & \textit{if unfilled} \\ a & \textit{if filled} \end{cases} \qquad \mathsf{surplus}(o) \coloneqq \begin{cases} 0 & \textit{if unfilled} \\ o - m & \textit{if filled with output } o \geq m. \end{cases}$$

Because filled outcomes carry the proof $o \geq m$*, the subtraction* $o - m$ *is well-defined in* $\mathbb{N}$ *(no underflow).*

Definition 9 ((A, B) pair)**.** *Define the objective for a single intent/outcome as*

$$\mathsf{AB}(o) := (\mathsf{volume}(o),\ \mathsf{surplus}(o)) \in \mathbb{N} \times \mathbb{N}.$$

For a batch (a finite list of intent/outcome pairs), define the batch score as componentwise sum:

$$\mathsf{AB}_{\text{batch}} := \sum_j \mathsf{AB}(o_j).$$

In Lean this is `batchAB`*, computed by a left fold with* `pairAdd`*.*

4.3 Lexicographic comparison

Definition 10 (Lexicographic order)**.** *For $p = (p_1, p_2), q = (q_1, q_2) \in \mathbb{N} \times \mathbb{N}$, define*

$$p \preceq_{\text{lex}} q \iff (p_1 < q_1) \ \vee \ (p_1 = q_1 \wedge p_2 \leq q_2),$$

and strict order $p \prec_{\text{lex}} q$ analogously with $p_2 < q_2$.

Proposition 4 (Total preorder)**.** *$\preceq_{\text{lex}}$ is reflexive, transitive, and total on $\mathbb{N} \times \mathbb{N}$, and the induced equivalence yields antisymmetry for $\preceq_{\text{lex}}$ on pairs.*

Proof. Standard case analysis on the first components; see Lean theorems:

- `lexLe_refl`
- `lexLe_trans`
- `lexLe_total`
- `lexLe_antisymm`

□

5 CPMM model and invariants

We model a constant-product pool with reserves $(x, y) \in \mathbb{N}^2$.

5.1 Zero-fee swap semantics

Definition 11 (Zero-fee output)**.** *Given reserves $(x, y) \in \mathbb{N}^2$ and input $a \in \mathbb{N}$, define the (zero-fee) output as*

$$q := \left\lfloor \frac{ya}{x + a} \right\rfloor. \tag{2}$$

The updated reserves are

$$x' := x + a, \qquad y' := y - q. \tag{3}$$

State-machine view. The swap semantics above define a deterministic transition function on states $(x, y) \in \mathbb{N}^2$ parameterized by an input $a \in \mathbb{N}$:

$$\mathsf{step}(x, y; a) := (x', y') \quad \text{where} \quad q = \left\lfloor \tfrac{ya}{x+a} \right\rfloor,\ x' = x + a,\ y' = y - q.$$

Equivalently, a sequential batch execution is just repeated application of step with inputs $a_1, a_2, \ldots$.

Lemma 1 (Output is reserve-bounded)**.** *For all $x, y, a \in \mathbb{N}$, the output satisfies $q \leq y$.*

Proof. Since $a \leq x + a$, we have $ya \leq y(x + a)$. Dividing by the positive denominator $x + a$ gives $\frac{ya}{x+a} \leq y$, and taking floors yields $q \leq y$. □

5.2 Worked example (single step)

To make the transition concrete, consider reserves $(x, y) = (100, 200)$ and input $a = 10$. Then

$$q = \left\lfloor \frac{ya}{x+a} \right\rfloor = \left\lfloor \frac{200 \cdot 10}{100+10} \right\rfloor = \left\lfloor \frac{2000}{110} \right\rfloor = 18,$$

so the updated reserves are

$$(x', y') = (110,\ 182).$$

The product increases from $K = 100 \cdot 200 = 20000$ to $K' = 110 \cdot 182 = 20020$. The exact gap identity predicts

$$r = (ya) \bmod (x + a) = 2000 \bmod 110 = 20,$$

and indeed $K' = K + r$.

5.3 The constant-product invariant and the K-gap

Definition 12 (Constant product)**.** *Define* $K(x, y) := x \cdot y$*.*

Theorem 1 (K-monotonicity)**.** *Under a zero-fee swap update* (2)–(3)*, the product never decreases:*

$$K(x', y') \geq K(x, y).$$

Proof (via the exact K-gap identity). We strengthen the claim by showing the exact difference is a Euclidean remainder. Let $d := x + a$ and $N := ya$. By Euclidean division (1), there exist $q, r \in \mathbb{N}$ with $N = dq + r$ and $0 \leq r < d$. By definition $q = \lfloor N/d \rfloor = \lfloor ya/(x + a) \rfloor$ which matches (2). Then:

$$\begin{aligned}
K(x', y') &= (x + a)\big(y - q\big) \\
&= dy - dq \\
&= dy - (N - r) \\
&= (x + a)y - (ya) + r \\
&= xy + r.
\end{aligned}$$

Thus $K(x', y') - K(x, y) = r \geq 0$. This is mechanized as `k_gap_exact` in `CPMMInvariants.lean`. □

Definition 13 (K-gap remainder)**.** *Define the per-swap remainder (the* K-gap*) as*

$$r(x, y, a) := (ya) \bmod (x + a). \tag{4}$$

Then the exact K-gap identity is

$$(x + a)\Big(y - \left\lfloor \frac{ya}{x+a} \right\rfloor\Big) = xy + r(x, y, a). \tag{5}$$

Corollary 1 (Characterizations)**.** *For all* $x, y, a \in \mathbb{N}$*:*

1. $K(x', y') \geq K(x, y)$ *(nonnegativity),*
2. $K(x', y') < K(x, y) + (x + a)$ *(bounded increase, since* $r < x + a$*),*
3. $K(x', y') = K(x, y)$ *iff* $(x + a) \mid (ya)$ *(zero remainder).*

Proof. Each follows from (5) and standard properties of mod. See `k_gap_nonneg`, `k_gap_bounded`, `k_gap_zero_iff`. □

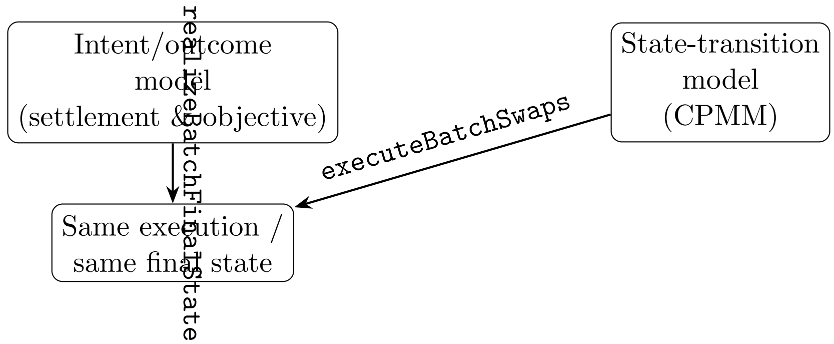

Figure 1: Two views of batch execution that agree by refinement: an intent/outcome view (used for settlement and objective computation) and a reserve state-transition view (used for CPMM invariants).

5.4 Batches: telescoping the exact K-gap

Definition 14 (Sequential execution)**.** *Fix an initial state* (x_0, y_0) *and a list of inputs* $[a_1, \ldots, a_n]$*. Define the sequence of states by iterating* (2)–(3)*:* $(x_i, y_i) \mapsto (x_{i+1}, y_{i+1})$ *using input* a_{i+1}*.*

Theorem 2 (Exact telescoping sum)**.** *For the sequential execution above,*

$$K(x_n, y_n) = K(x_0, y_0) + \sum_{i=0}^{n-1} r(x_i, y_i, a_{i+1}), \tag{6}$$

where r *is the per-step remainder* (4)*.*

Proof. Induction on n. The induction step applies the single-step identity (5) at state (x_i, y_i) and accumulates remainders. This is mechanized as `executeBatchSwaps_K_gap_sum`. □

Corollary 2 (Batch K-monotonicity)**.** *Since each remainder is nonnegative,* $K(x_n, y_n) \geq K(x_0, y_0)$ *for every batch.*

6 Unification: settlements, objective, and CPMM state

There are two complementary models for batch execution:

1. **Intent/outcome (settlement & objective) model.** A batch is a list of pairs (intent, valid outcome). It induces: (a) an aggregated settlement in `Settlement` (for Δ-analysis) and (b) an aggregated objective score in $\mathbb{N} \times \mathbb{N}$ (for optimization).

2. **State-transition (CPMM) model.** A batch is a list of inputs $[a_1, \ldots, a_n]$ executed sequentially on reserves, yielding a final reserve state with verified K properties.

6.1 Aggregating settlements

For each intent/outcome pair, the Lean development defines a per-pair settlement: unfilled contributes 0, and filled contributes $\mathsf{swap}(a, o)$. The batch settlement is the fold (sum) of these settlements. Using the homomorphism property, scalar net-flow composes:

$$\Delta\Big(\sum_j s_j\Big) = \sum_j \Delta(s_j).$$

Consequently, if each filled execution satisfies $a \geq o$ (a *same-unit* condition), then Δ-safety composes over the batch (`batch_safe_implies_settlement_safe`).

6.2 Refinement bridge and unified safety theorem

The key bridge is that the output-generating function (computing q_i sequentially) reaches the same final state as the pure state-transition fold:

$$\mathsf{realizeBatchFinalState}(s, \vec{a}) = \mathsf{executeBatchSwaps}(s, \vec{a}).$$

This is proved as `realizeBatch_final_state` in `BatchCPMMUnification.lean`.

As a consequence, the CPMM invariants proven for the state-transition model apply to the same execution that feeds settlement/objective computations. The combined statement is captured by `unified_batch_safety`, which simultaneously establishes:

1. $K(\text{final}) \geq K(\text{initial})$,
2. $K(\text{final}) = K(\text{initial}) + \sum \text{remainders}$ as in (6),
3. reserve-in accounting: $x_{\text{final}} = x_{\text{initial}} + \sum_i a_i$.

7 Mechanization notes

The results above are mechanized in Lean 4/Mathlib (no axioms and no `sorry`). To typecheck/build the Lean project:

```
cd lean-mathlib
lake build
```

8 Limitations and future work

This development is deliberately minimal and focuses on arithmetic correctness and composition. Several important directions remain:

- Multi-asset generalization: settlements as $\mathbb{Z}^n$ vectors; richer notions of value conservation.
- Full batch clearing optimality: argmax existence over finite candidate sets and deterministic tie-breaking.
- Multi-hop routing and cross-pool execution: lifting K-style invariants across routing graphs.
- Overflow-aware models and byte-level constraints matching on-chain execution environments.
- Expanded fee models and protocol-specific mechanics (dynamic fees, rebates, TWAP/oracle guards, etc.).

References

[1] Euclid. Elements (Book Vii): Euclidean division. *Classical source*, 300. Standard statement: for
N, d
in
$mathbbN$ with $d > 0$, there exist unique q, r s.t. $N = dq + r$ and 0
$ler < d$.