When Is a Smart Contract Specification Correct, and How Can We Evaluate It?

Derek Sorensen

14 June 2025

Premises

Formal verification is:

1. Sold as: proving the code "free of vulnerability"
2. Actually: correct w.r.t. a spec
3. Useless if the specification is incorrect

Premises

Formal verification is:

1. Sold as: proving the code "free of vulnerability"
2. Actually: correct w.r.t. a spec
3. Useless if the specification is incorrect

Focused case study: Smart contracts

Smart Contracts

A **smart contract** is a program that executes on a blockchain.

Key components:

1. Entry points
2. Storage
3. Native token balance

Key features:

1. Often manages money
2. Implement a market or game
3. Ideal for FV

ERC20 Token

1. Entry points:
   - `transfer`
   - `mint`
   - `burn`
2. Storage:
   - `balances`
3. Native token balance:
   - zero

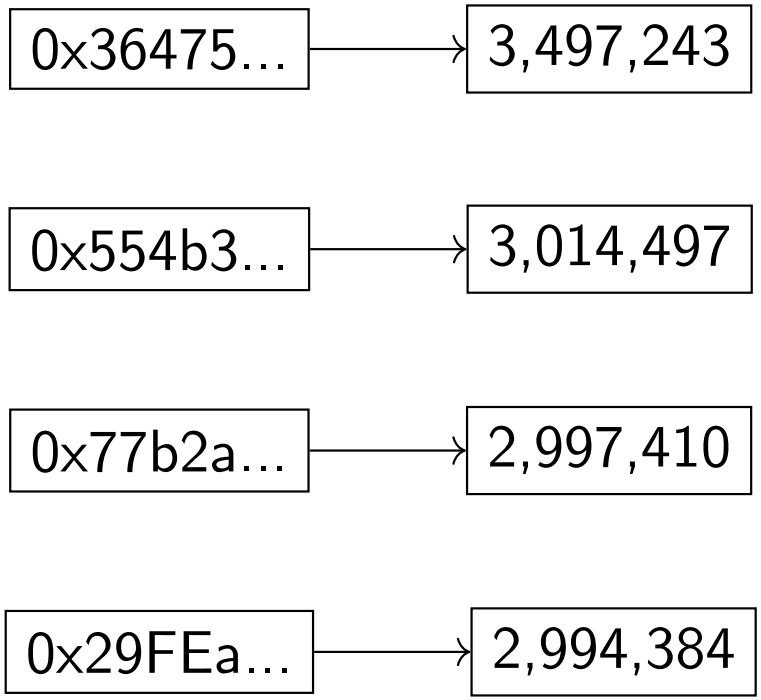


Figure: A snapshot of USDC balances on Ethereum mainnet.

Logic of a Token Contract

Complex specification logic:

- algorithmic stablecoins
- synthetics
- elaborate tokenomics

Specification of an AMM

1. Entrypoints:
   - `swap`
   - `add_liquidity`
   - `remove_liquidity`
2. Storage:
   - `lp_token_balance`
   - `token_x_balance`
   - `token_y_balance`
3. Native token balance:
   - Possibly nonzero (treasury)

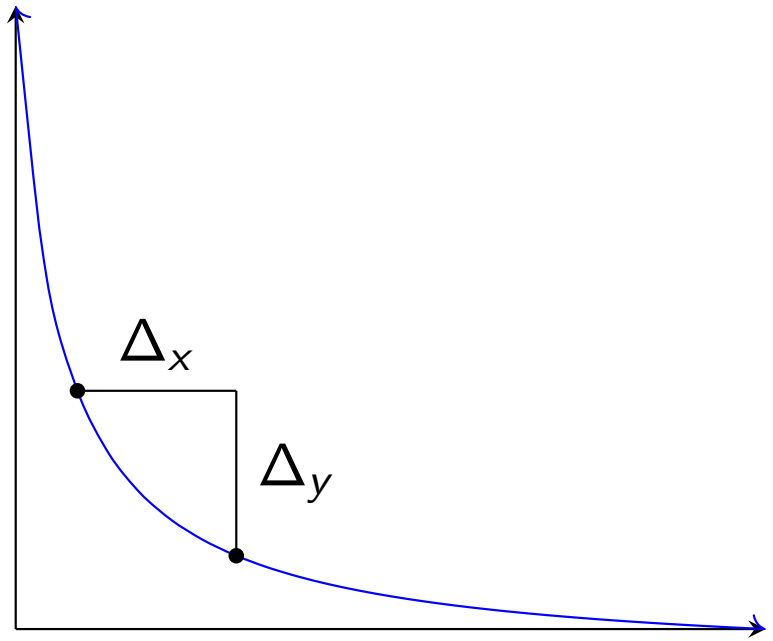


Figure: A trade of Δ_x for Δ_y along the indifference curve $xy = k$.

Specification of an AMM: A Translation

Desired Behavior

1. Efficient price discovery
2. Incentivized liquidity provision
3. Reasonable (?) slippage

vs.

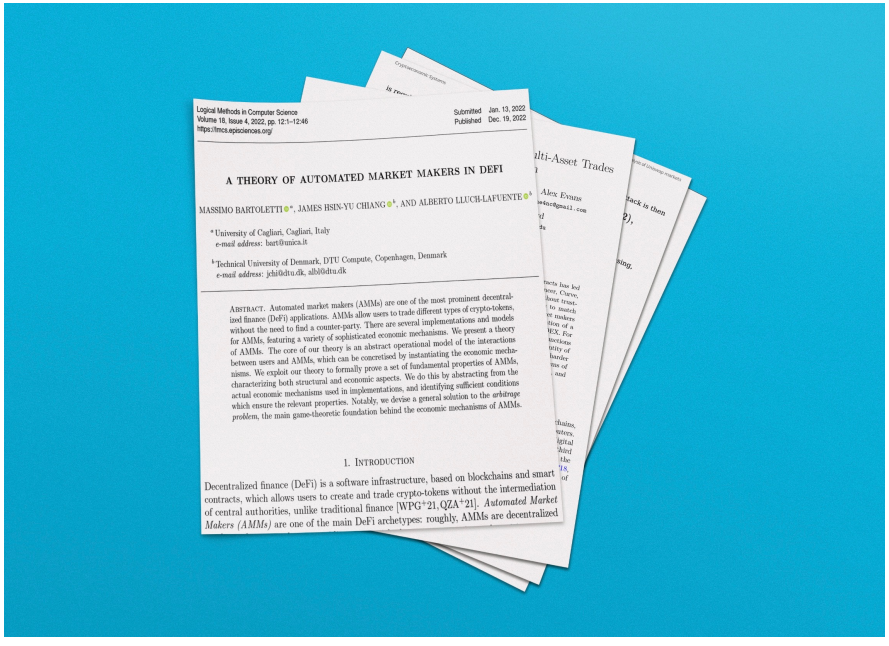
Formal Specification

1. Entrypoint specification
   - `swap`
   - `add_liquidity`
   - `remove_liquidity`
2. Storage specification
   - `lp_token_balance`
   - `token_x_balance`
   - `token_y_balance`

Formally Derived Desirable Properties of AMMs

Formally derived properties of AMMs

1. Demand sensitivity
2. Nonzero prices
3. Swap rate consistency
4. Zero-impact liquidity change
5. Arbitrage sensitivity



Translation Process

High-Level Intuition $\leftrightsquigarrow$ Formal Specification

Translation Process

High-Level Intuition $\leftrightsquigarrow$ Formal Specification

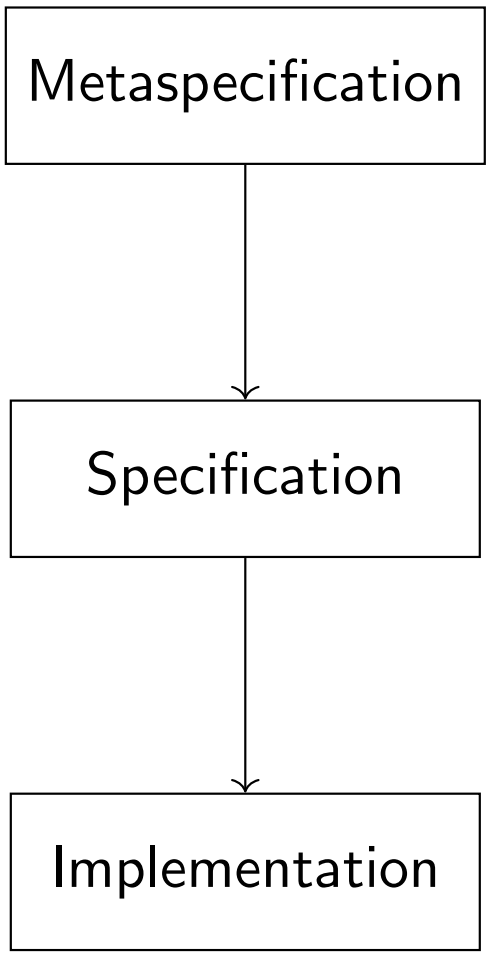
"Meta work"

- Metaspecification
- High-level properties in prose

Mathematical abstractions

- Morphisms of contracts
- Isomorphisms of contracts

Specification/Metaspecification



Metaspecification

- High-level, cryptoeconomic properties

Specification

- Entrypoints, storage, *etc.*

Implementation

- Structured pool contract

Structured Pools

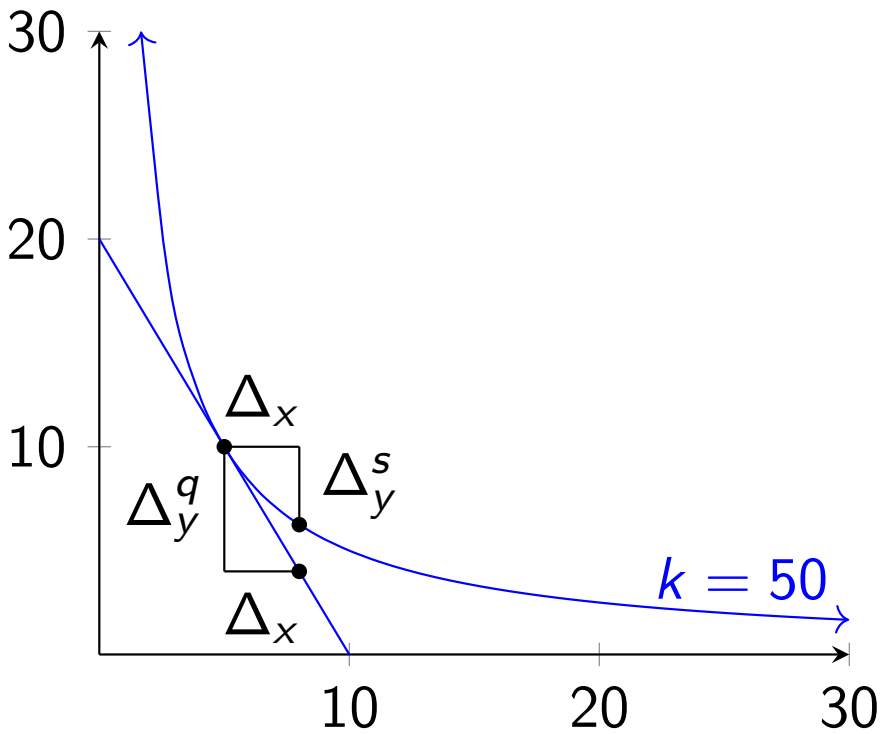


Figure: A trade of $\Delta_x = 3$ for Δ_y^q and Δ_y^s, respectively, at $k = 50$. $\Delta_y^q = p_q \Delta_x$ is the trade priced at the *quoted price* p_q and $\Delta_y^s = p_s \Delta_x$ is the trade priced at the *swap price*

Entrypoints:

- `DEPOSIT`
- `WITHDRAW`
- `TRADE`

Storage:

- Exchange rates
- Balances

$$r_x' := \frac{r_x x + r_y \Delta_y}{x + \Delta_x}$$

Structured Pools

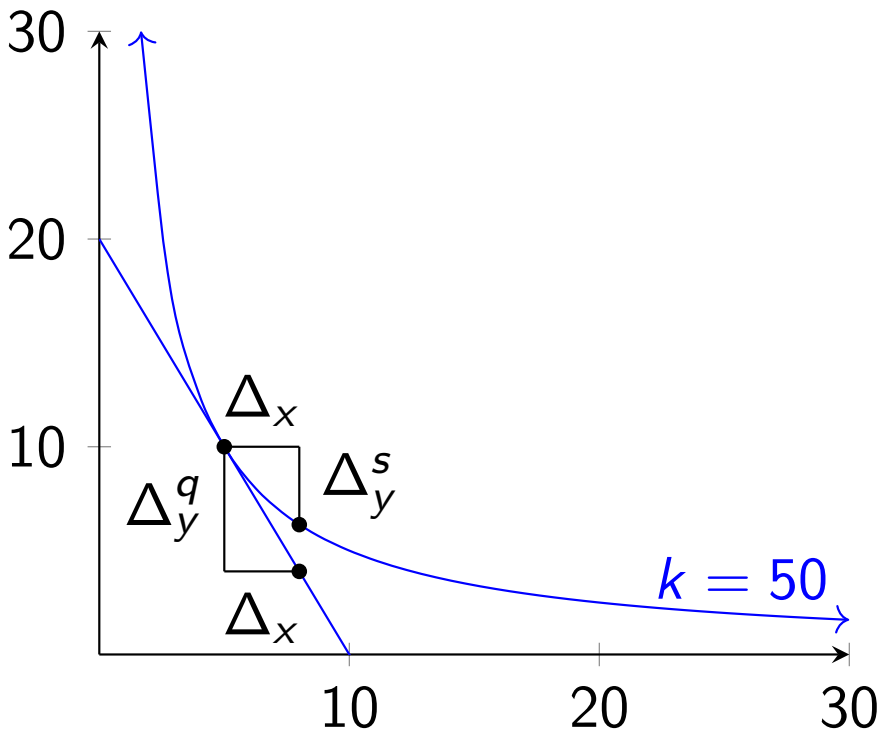


Figure: A trade of $\Delta_x = 3$ for Δ_y^q and Δ_y^s, respectively, at $k = 50$. $\Delta_y^q = p_q \Delta_x$ is the trade priced at the *quoted price* p_q and $\Delta_y^s = p_s \Delta_x$ is the trade priced at the *swap price*

The metaspecification:

1. Demand sensitivity
2. Nonpathological prices
3. Swap rate consistency
4. Zero-impact liquidity change
5. Pooled consistency

Learnings From a Metaspecification

1. We have a formal definition of a "correct" specification
2. The metaspecification informs choices made in the specification

$$r'_x := \frac{r_x x + r_y \Delta_y}{x + \Delta_x}$$

From here: formalize theories, *etc.*

Evaluation

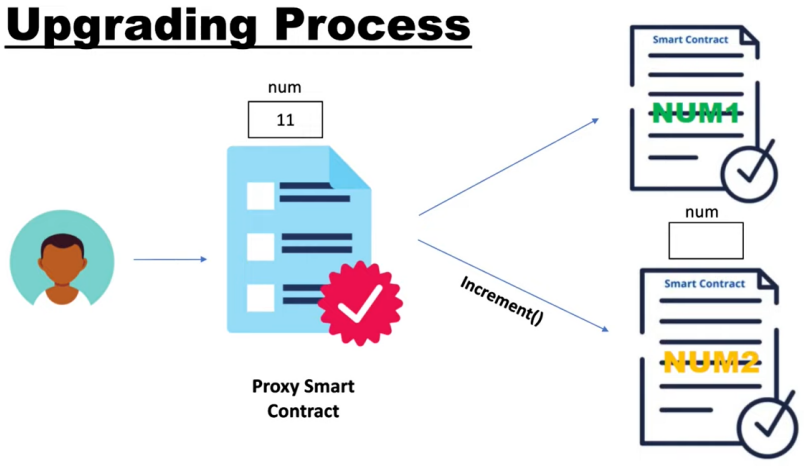
Goal: Intuition $\leftrightsquigarrow$ Formal specification

Tool: Metaspecification

Application: Economic properties of contracts

- We can state the desired properties

Contract Upgrades



Setting: Contract upgrades

Nomad, Uranium Finance, NowSwap

1. $>$ \$190 mn lost
2. Failed to align with the intent of the upgrade

Contract Morphisms

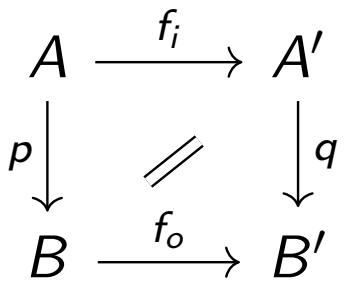


Figure: A morphism from p to q.

Contract Morphism: a commutative square between contracts (as pure functions).

Mapping:

- Inputs to inputs
- Outputs to outputs

Such that the square commutes.

Backwards Compatibility (C_old → C_new)

C_old

```
Inductive entrypoint1 :=
  | incr (u : unit).
```

C_new

```
Inductive entrypoint2 :=
  | incr' (u : unit)
  | decr (u : unit).
```

```
Definition msg_morph (e : entrypoint1) : entrypoint2 :=
    match e with | incr _ ⇒ incr' tt end.
```

C_old → C_new specifies *in what way* C_new is backwards compatible with C_old with an **embedding**.

Swap Precision Upgrade (C_old ⟵ C_new)

C_old	C_new
get_bal : storage → N.	get_bal : storage → N.
trade : trade_data → e_msg.	trade : trade_data → e_msg.
calculate_trade	calculate_trade_precise

C_old ⟵ C_new is a proof that C_new behaves like C_old, up to rounding with a **quotient**.

Optimization (C_old ≅ C_new)

C_old

```
Inductive entrypoint :=
  | addOwner (a : N)
  | removeOwner (a : N)
  | swapOwners (a_fst a_snd : N).
```

```
Record storage_arr :=
  { owners_arr : list N }.
```

C_new

```
Inductive entrypoint :=
  | addOwner (a : N)
  | removeOwner (a : N)
  | swapOwners (a_fst a_snd : N).
```

```
Record storage_ll :=
  { owners_ll : FMap N N }.
```

C_old ≅ C_new indicates that the optimization was done correctly with an **isomorphism** (bisimulation).

Porting Proofs Over Morphisms

($\rightarrow$)

- Compression of computational threads
- Properties port over the morphism, modulo the compression

($\leftarrow$)

- Embedding of computational threads
- Properties port over the morphism, modulo the embedding

($\cong$)

- Establishes a correspondence of computational threads
- Properties port over the morphism, modulo the equivalence

Evaluation

Intuition $\leftrightsquigarrow$ Formal Specification

Tools:

1. Metaspecification (the "meta")
2. Morphisms (abstractions)

Both are ideally set in a theorem prover.