d$\mathcal{F}$usion Exchange Decentralized Multi-Token Batch Auctions as a Snark-Application

Benjamin H. Smith

GNOSIS

February 25, 2019

Joint work with Alex Herrmann, Felix Leupold, Oliver Beige & Tom Walther

Overview

1. d$\mathcal{F}$usion
   - Multi-Token Batch Auction
   - Benefits of this model
2. Scalability & Decentralization
   - Achieving Decentralization
   - Achieving Scalability
3. Snark Applications
   - Rudimentary Components
   - Event Listener
   - Contract Driver
   - Snarks revisited
4. Upcoming Challenges

We are going to take a top-down approach

- *Limit orders* between any (registered) token pairs are collected in a batch (over 3 minutes or up to N orders).
  N - dictated by the capacity of the snarks†
- *Order matching* algorithm is modelled as a *Mixed Integer Program* with;
  - *Objective Function* is one of trader's welfare or trading surplus
  - *Feasibility Region* is encompassed by
    1. Respected Limit Prices
    2. Conservation of Value [Tokens not created or destroyed]
    3. Price Coherence [$p_{ij} \cdot p_{ji} = 1$]
    4. Arbitrage Freeness [prices along cycles multiply to 1]

Benefits

- *Ring Trades* - higher likelihood or order fulfilment
- *Inherently fair* - as defined by the feasible region

Decentralization

Placing and settlement of orders occurs in an Ethereum Smart Contract

- Anyone can submit solution proposals for auction results
  - Smart Contract will choose the best
  - Reward mechanism for best solution
  - Winning solution will be expected to provide Snark Proof (Proof of Optimization)
- Anyone can propose state transition
- State transitions can be challenged

Scalability

Limited but sufficient on-chain storage

- K - accounts
- T - tokens
- constants (max tokens, max accounts, etc)
- Account state hash (representing balances $\{B_{k,t}\}_{\forall k,t}$)

Atomic Swaps

- auction settlements
- other states transitions (deposits & withdrawals)

SNARKS

(Succinct Non-interactive ARguments of Knowledge)

- **Prover** (a.k.a. Solution Proposer) does computation off-chain to prove that auction results are feasible.
- **Verifyer** snark-proof is submitted on-chain in the form of a Smart Contract
- Snarks used for all State Transitions (i.e. account balance updates)
  1. Processing Deposits
  2. Processing Withdrawals
  3. Auction Settlement

Smart Contract

Contract contains

1. Elementary Items and Accessibility
    - tokens, accounts and registration of them
2. Participation requests
    - deposit, withdraw and limit order
    - Emit Events
3. State Transitions (balances)
    - processing deposits, withdrawals and auction results
4. Challenge, Resolve & Rollback

Off-chain

- Requests are emitted as an event by the Contract
  - **Deposit**
    "Account k deposited d of token t"
  - **Withdraw**
    "Account k withdrew d of token t"
  - **Limit Order**
    "Account k to trade $\leq d$ of t_i for t_j if exchange rate is $\leq r$"
- Contract doesn't store the information contained in events.
- Event Listener collecting and storing relevant info (Anyone)

Off-chain

Information stored by listener is used to "Drive" the contract

- Perform all the balance updates and hashing
- Call process-request functions (AccountStateTransitions)
- (On Challenge) Computes snark proofs

deposits & withdrawals

Snark contains all information regarding deposits in that slot

Auction Settlement

Snark contains, prices and limit-order fulfilment is sufficient to demonstrate constraints of Linear Program

Future Features

1. Fork-able States
2. Basket Orders
3. Batch Requests (i.e. offchain order collection - as a service)
4. Continuation Orders

Resources

All our efforts are open-sourced through Gnosis GmbH at

http://github.com/gnosis/

- *Formal Specification*: gnosis/dex-research
- *Smart Contracts*: gnosis/dex-contracts
- *Infastructure (Listener & Driver)*: gnosis/dex-services