CUP PERPS TECHNICAL DOCUMENTATION (V0.1)

ZYGOMEB

1. High Level Overview

The idea of Cup Perps is to take the concept of Perpetual Swaps and take a look at from a more streamlined way, one that more accurately represents the demand/supply dynamics of going long/short, making sure that they're equally motivated and able to facilitate a dynamic risk/reward environment.

At current moment the idea is given as-is, in a raw state, as due to time constraints economic analysis of the protocol was not possible. It is an experimental model; a very different take on how perpetual swaps could be structured.

Each market consists of two liquidity pools, which I refer to as **LongCup** and **ShortCup**, each pooling liquidity in a single chosen token. The goal of the system is to encourage liquidity in both cups to be equal, by giving rebates to the smaller cup on any price change (lose less and win more for smaller cup, inverse for bigger cup).

A data feed **Price(T)**, that represents the asset pair that is synthetically bet on in this derivative product. Unlike in traditional perpetual swaps we do not need to worry about trying to make the derivative trade on par with underlying as the system tracks only in delta-price changes. At any point, buying in is equivalent to getting exposure correlated with the emulated asset (re-scaled via any cup imbalance).

The system works with any frequency of data point updates, and in an idealized model, updates given as often as possible (before every trade execution).

2. Technical Specification

At every discrete point in time **T**, we have an associated oracle (true) price feed **Price(T)**. Within the system's memory we store the last price feed, **PriceLast**, as well as the time at which it was last updated **Last**. The two liquidity pools, cups, **LongCup** and **ShortCup**, as well as their associated liquidity provision tokens, **LongLp**, **ShortLp**. Other system parameters are: **Leverage**, **FundingCoeff**, explained shortly.

The **update()** function:

```
assert(T > Last)
assert(Price(T) > 0)

if Last == Price(T) { return }

Delta = (Price(T) / PriceLast - 1) * Leverage
LongD = Delta * LongCup - LongCup
ShortD = Delta * ShortCup - ShortCup

Funding = min(ShortCup/LongCup, LongCup/ShortCup) * FundingCoeff
AdjDelta = if |LongD| > |ShortD|
    then
        |ShortD| * if Delta > 0 then Funding else 1/Funding
    else
        |LongD| * if Delta < 0 then Funding else 1/Funding

if Delta > 0 {
    transfer AdjDelta from ShortCup to LongCup
} else {
    transfer AdjDelta from LongCup to ShortCup
}

PriceLast = Price(T)
Last = T
```

The **deposit(Input))** function:

```
Choosing the appropriate Long/Short version of variables:

update()

assert(Input.type == Cup.type)
LpCaller = Lp * ((Cup + Input) / Cup - 1);
Lp += LpCaller;

transfer Funds to Cup
mint LpCaller to Caller
```

The **withdraw(Input** function:

```
Choosing the appropriate Long/Short version of variables:

update()

assert(Input.type == Lp.type)
Payout = Input / Lp * Cup;
Lp -= Input;

burn Input
transfer Payout to Caller
```