SynFutures@v1 Technical Paper

SynFutures Team

info@synfutures.com

December 02, 2020

1 Overview

Inspired by the success of the decentralized spot exchange Uniswap, SynFutures is designed to create a decentralized synthetic asset derivatives trading platform and launches its first version with a Futures trading market.

- Similar to Uniswap where anyone can create a new spot trading pair, SynFutures@v1 allows any liquidity provider to create digital assets Futures trading pairs with arbitrary assets and arbitrary expiration dates.
- SynFutures@v1 adopts an "sAMM (Synthetic Automated Market Maker)" model, where the asset pricing follows a Constant Product Formula, while for liquidity provision, only one single asset (Quote Asset) of the trading pairs is required to be supplied, and the other asset (Base Asset) is automatically synthesized by the contract, thus supporting not only Ethereum native assets, but also cross-chain and real-world assets such as gold and foreign currencies.
- SynFutures@v1 also introduces the concept of "ALQ (Automated Liquidator)" , who passively performs the role of the liquidator by providing liquidity only and earns trading fees. In the first version, sAMM is the liquidator at the same time. With that, for the first time in DEX to our knowledge, partial liquidation of a trader' s leveraged position becomes possible.

2 Oracle

The price of the Futures normally tracks that of the underlying spot price. Subject to the development of Oracle in the blockchain industry, not all Base/Quote asset pairs (As an example, in the trading pair ETH/USDT, ETH is the Base asset, hereinafter referred to as "Base", USDT is the Quote asset, hereinafter referred to as "Quote") have reliable Spot prices Oracle.

SynFutures@v1 currently only introduces Uniswap and Chainlink as Oracle and requires Quote to be a native asset on Ethereum. The assets that can be used as Quote when SynFutures@v1 launches include: ETH, USDC, USDT, and DAI.

- If Base is also a native asset on Ethereum, SynFutures@v1 uses Uniswap as the price Oracle by default.
- If Base is not a native asset on Ethereum, SynFutures@v1 uses Chainlink as the price Oracle by default.

One point to note is that on Chainlink, BTC and other assets are quoted in USD, while the pricing of USDC, USDT, DAI, and other USD stablecoins on Ethereum fluctuates in a small range relative to USD. For simplification, SynFutures@v1 treats 1USDC = 1USD and uses USDC as the USD pegged asset on Ethereum for Futures contracts relying on Chainlink Oracle. That is, the Quote asset in the futures contracts that rely on Chainlink's USD-related pricing in SynFutures@v1 will all be USDC instead. For example, the Quote asset in the BTC/USD Futures contract in SynFutures@v1 is actually USDC. If the BTC/USD pricing fed by Chainlink is 18000, the BTC/USD futures contract of SynFutures@v1 treatsthe current pricing of BTC as 18000 USDC.

Notably, SynFutures@v1 also introduces a special handling mechanism for Oracles. For example, Oracle using Uniswap's spot index is vulnerable to price manipulation. SynFutures@v1 imposes additional restrictions to mitigate the risk, which will be described in detail later.

3 Market Making and Trading

SynFutures@v1 introduces "sAMM" (Synthetic Automated Market Maker) model as the Market-Making mechanism. The sAMM provides liquidity for Futures contracts and always makes prices to traders based on the Constant Product formula $xy = k$.

Similar to Uniswap, a Liquidity Provider (LP) can create a new pool with and provide liquidity to arbitrary asset pairs, and with one more arbitrary parameter for Futures contract - expiry dates. While Uniswap requires LP to provide two assets of a pair into the liquidity pool at the same time to ensure atomic physical settlement, this is not the case for Futures contracts since net settlement in one asset of the pair is sufficient.

Therefore, to provide liquidity to a Base/Quote asset pair, with sAMM model of SynFutures@v1, an LP will transfer the Quote asset token only to the sAMM, among which half provided is used as Quote asset, and the other half to synthesize the position of the Base asset, that is, to create a 1x long position of the Base Asset for this futures contract. Since LP originally held Quote assets only and did not have any exposure to the price risk of Base assets, the sAMM contract will at the same time allocate a short position of the same size as the newly created long position to the same user to hedge this risk. Thus the action of adding liquidity to the sAMM does not change the total risk profile of the liquidity provider at the start, as the newly created long and short positions offset each other. However, after adding liquidity to sAMM, the liquidity provider has also become a trader due to the short hedging position, and needs to maintain sufficient margin in the account to meet the margin requirement.

A trader has a corresponding Account in the futures contract he or she participates in, and AMM also has its own Account. The Account records the current user's position, account balance, and other information. Users can deposit funds (deposit()) into and withdraw (withdraw()) from their accounts. After

depositing into the account, the fund can be used as margin to open a long or short (trade()) position with AMM, while LP can use the funds to add liquidity (addLiquidity()). Assets used to provide liquidity can also be removed (removeLiquidity()). When liquidity is added, LP will get the corresponding LP Token, and when liquidity is removed, the corresponding LP Token will be burned. When Futures price fluctuates and the margin balance of an account for a trader is not sufficient to meet the maintenance margin requirement of its current position, anyone (except the trader account itself) can initiate the liquidation of the account (liquidate()). Liquidation usually means that the liquidation initiator uses its own account to take over the position of the target account, which requires a certain amount of margin in the liquidation initiator' s account. In order to lower this barrier, SynFutures@v1 has introduced an "Automated Liquidator" mechanism: Liquidate the target account with the help of AMM (liquidateByAmm()), that is, the liquidation initiator can use the liquidity in AMM's account to liquidate the target account and receive rewards.

4 Contract Life Cycle

Drawing from traditional and centralized market practices, SynFutures@v1 divides the life cycle of a Futures contract into three stages: TRADING, SETTLING, and SETTLED.

A futures contract enters the TRADING state when newly created and initialized, and later enters the SETTLING state (at least one hour), after which the SETTLED state. It is worth mentioning that forcing a futures contract to be in SETTLING state for at least one hour may lead to the actual expiration time of the futures contract later than the specified time when the contract was created. This is because the status update of the smart contract itself can only be triggered by a transaction. Specifically, if no transaction occurs about an hour before the specified expiration time, the futures contract will fail to enter the SETTLING state strictly one hour before the predefined expiration time.

To cope with this situation and enable the contract to be in SETTLING state at least one hour before expiration, SynFutures@v1' s design is as follows: When processing transactions in the TRADING state, the smart contract will first determine whether the Futures contract should be switched to SETTLING (even SETTLED) state at this time based on the preset expiration time. If so, the contract state would be changed to SETTLING, and the expiration time would be reset to be one hour later than the current block time. With that, one could be ensured that all contracts will go through at least one hour of SETTLING state before expiration (As mentioned, the SETTLING state of futures contracts may last longer than one hour). It can be expected that the actual expiration time of an actively traded Futures contract will be fairly close to the specified expiration time when the contract was created. At the same time, SynFutures@v1 introduces an additional reward mechanism in order to encourage users to update the state of Futures contracts by initiating transactions.

In the TRADING and SETTLING states, the futures contract mark price is calculated differently: the SETTLING state uses a time-weighted moving average to calculate the mark price, which will be described in detail later. In addition, under different contract states, the operations that the trader and LP can perform, and the respective restrictions are different:

- In the TRADING state: the trader can deposit(), withdraw(), update(), trade(), liquidate() and liquidateByAmm(), while the LP can deposit(), withdraw(), addLiquidity(), and removeLiquidity() Of course, LP can at the same time play the role of the trader and also performs permitted operations.
- In the SETTLING state: all operations can only reduce, but not increase the position in the corresponding account. Therefore, a trader can perform the various operations in the TRADING state but with additional constraints: when trading, a trader can only close outstanding positions but not increase or open new positions; and LP is allowed to removeLiquidity() only but not addLiquidity(), since adding liquidity will create a hedge position at the same time and thus increase the position in the AMM account.
- In the SETTLED state: both the trader and the LP can claim all the fund via settle(). settle() will firstly withdraw assets for LP by burning all the remaining LP tokens, then close all position according to the settlement price. Note that users' margin will be automatically transferred from the Account to the user' s own address when settle() is performed.

In addition, SynFutures@v1 also introduces a state named EMERGENCY in order to deal with unforeseen abnormal conditions on chain (such as Oracle failures, etc). In the EMERGENCY state, normal users and LPs are not allowed to perform any operations, and the system administrator will guide the futures contract into the SETTLED state with a fair and reasonable settlement price.

5 Glossary

- Initial Margin Ratio (IMR): minimum initial amount of margin as a ratio to the position value to open a new trade
- Maintenance Margin Ratio (MMR): minimum amount as a ratio to the position value a trader should maintain in the margin account after opening a trade, below which the account could be liquidated
- Insurance Premium Ratio: penalty ratio charged to an account when its position is liquidated, the penalty is sent to insurance fund
- Index Price: price of the underlying spot trading pair as supplied by Oracles
- Mark Price: price used to mark to market positions of all accounts, determine whether a futures position should be liquidated and the settlement price at expiry
- Fair price: current market price as implied by the AMM inventories
- Available Margin: margin amount that can be withdrawn or used to open new positions
- Entry Cost or Entry Notional: entry cost of the current position in the account
- Account profit and loss (PNL): unrealized profit and loss of the current position as of its entry, valued against mark price

- Exponential Moving Average (EMA): a technic used to smooth the change in mark basis, please refer to Wikipedia[1]
- Social Loss: losses incurred due to liquidation for one side of the trade that is shared by the entire counterparty side
- Time Weighted Average Price (TWAP): average price based on time weights
- Open Interests: number of all open positions of the same side. Note that open interest of long side is always the same as the open interest of the short side

6 Margin and Account

Futures contracts with the same Base, Quote, but different expiry rely on different sAMMs (fixed margin), which could lead to relatively dispersed liquidity. The second version of SynFutures will use a new sAMM model to solve this problem and implement Cross Margin. And for the first version, when launched, the expiration time of all futures contracts will be aligned to 8am UTC time on every Friday of the week where the expiration time specified by the user is located. The restriction on the mandatory expiration time alignment will be relaxed at an appropriate time and the expiration time of all futures contracts will be aligned to the user-specified expiration date at 8 am UTC time, subject to the development of the market and could be varied by different trading pairs with different trading volumes and requirements.

In view of the current development of on-chain Oracle, SynFutures@v1 initially supports the following assets to be used as Quote token: ETH, USDC, USDT, and DAI.

SynFutures@v1 maintains the margin configuration information MarginParam in the global parameters for each supported Quote token. The structure is defined as follows. Among them, updateReward specifies the reward amount when the user updates the corresponding futures contract status.

```
struct MarginParam {    // only takes 1 slot
    bool allowed;       // margin can be used as Quote?
    bool alignToFriday; // align expiry to Friday?
    uint128 updateReward; // reward in margin's decimal to incentive user to update futures
}
```

The futures contracts in SynFutures@v1 are uniquely determined by Base, Quote, and Expiry, each with its own sAMM to provide liquidity, and also a corresponding account list to record the balance, position, and entryNotional etc. of the trader, LP, and sAMM.

```
struct Account {
    int128 balance;
    int128 position; // positive for LONG, negative for SHORT
    uint128 entryNotional;
    int128 entrySocialLoss;
}
```

SynFutures@v1 uses the structure Account to record the on-chain status of each account:

[1] WikiPedia - Exponential smoothing. https://en.wikipedia.org/wiki/Exponential_smoothing

- balance: represents the account balance of the current account. Since the trading loss may exceed the fund balance of the account, this field may be a negative value;
- position: indicates the long or short position of the current account, a positive number means LONG, and a negative number means SHORT;
- entryNotional: is the entry cost of the current position, that is, the entry price times quantity;
- entrySocialLoss: represents the social loss of the current position direction (LONG or SHORT)

During the entire life cycle of a futures contract, multiple accounts might be liquidated. For example, when the price of Base relative to Quote continues to rise, the account holding a SHORT position will continue to suffer a loss. When AccountBalance + UnrealizedPnl $<$ Position $\times$ MarkPrice $\times$ MMR, The account is no longer safe and can be liquidated, and liquidation may result in a negative balance of the account.

- When the balance of the account becomes negative resulting from liquidation, the insurance fund of the futures contract will be firstly used to reward the liquidator, and cover the shortfall,
- If the amount of the deficit exceeds the insurance fund balance, the excess loss will accumulate in the socialized loss in the LONG direction, and the loss will be shared in proportion to the positions held by all holders of the LONG side at this time.

In order to calculate the social loss that the account should bear during the period from opening the position to the liquidation of an account, it is only necessary to know the difference between the accumulated social loss at the two moments and the user' s positions. Therefore, it is necessary to save the cumulative social loss of the user's position holding in the Account.

The key to the feasibility of the entire futures contract is the design of sAMM. Assets representing Base's LONG or SHORT positions do not need to be physically delivered, so sAMM of SynFutures instead synthesizes the Base positions. When the futures contract is created, the expiration time will be aligned according to the configuration information in MarginParam. After the futures contract is created, the LP needs to inject liquidity into the pool to complete the initialization of the futures contract. Of course, before injecting liquidity, you first need to deposit Quote assets into the contract. To make LP' s operations easier, SynFutures@v1 provides two additional methods for LP: depositAndInitPool(), which will combine the operations of deposit, initializing a new pool and adding liquidity, and depositAndAddLiquidity(), which will combine the operations of deposit and adding liquidity.

7 Numerical Precisions

The numerical precisions of different assets on Ethereum are different. In order to facilitate unified processing of the contract, SynFutures@v1 internally aligns the precision of all assets to 18. This implementation also affects the input parameters of the contract methods for user interaction, and SynFutures@v1 does not support assets with precision higher than 18 as Quotes. For all the fields in SynFutures@v1

requiring input of the asset amount, the corresponding input value is amplified by default. For example, for USDC with a precision of 6, 1000×10^6 means 1000USDC, and the same value is recorded in SynFutures@v1 as 1000×10^{18}, and SynFutures@v1 will convert the precision when withdraw() method is performed: when interacting with the corresponding asset contract, the value will be converted to appropriate precision, and the number of assets less than 1 minimum unit will be directly truncated.

In terms of numerical precision for value, SynFutures@v1 also uniformly enlarges all other values by 10^{18}, except for seconds representing time. For example, the precision of balance in Account is with a precision of 18 according to the aforementioned style. position, entryNotional, and entrySocialLoss are also enlarged by 10^{18}. This on one hand is to circumvent the limitation that Solidity not supporting floating-point numbers: SynFutures@v1 considers integer values smaller than 10^{18} are decimals, for example, the value of 0.1 in 10^{18} corresponds to 10^{17}; On the other hand, is to improve precision: floating-point number calculation will always suffer from precision loss while rounding to the lowest digit after amplification can minimize the impact of errors.

There is one exception for numerical precision, which is the storage of global parameters. Since most of the global parameters represent a certain percentage, that is, the value is a floating-point number less than 1, the 4 precision decimal is sufficient for parameter configuration, and various interactions between users and futures contracts require access to multiple global parameters. In order to reduce the storage space on-chain and reduce gas consumption when the user interacts with the contract, the storage of global parameters is only enlarged by 10^4, which is enough to represent a 4-digit precision floating-point number. Additionally, uint16 can also be used to represent a configuration parameter, so that all global parameters can be stored in a slot on the Ethereum chain. Thus, when SynFutures@v1 uses global parameters internally, the global parameters will first be enlarged by 10^{14}, so that the corresponding value is consistent with the internal value of the system.

A natural question is, is int128 or uint128 enough to store the amplified value? The value range of int128 is $[-2^{127}, 2^{127} - 1]$, of uint128 is $[0, 2^{128} - 1]$, and $\log(2^{127} - 1, 10) \approx 38.23$, you can see that int128 and uint128 are sufficient for storing values. SynFutures@v1 internal operations are all expanded on uint256 or int256 type values. In order to prevent the multiplication of the enlarged value from causing the value to overflow, when two enlarged values are multiplied, the result will be reduced by 10^{18} and rounded to the lowest digit. Similarly, when dividing two enlarged values, the dividend will first be enlarged by 10^{18}, and then divisible and truncated according to the principle of rounding.

8 Synthetic AMM

8.1 Add/Remove Liquidity

After the futures contract is created, the LP needs to deposit and inject liquidity into the contract through the depositAndInitPool() method:

```
// deposit margin and initialize the pool, margin token amount and leverage in 10^18
function depositAndInitPool(
    uint wadAmount, uint initPrice, uint leverage, uint deadline
) public payable returns (uint)
```

The input parameters are as follows:

- wadAmount represents the amount of margin to be deposited. Note that this is the amount of margin assets represented by 18-digit precision.
- initPrice is the price of the specified Base/Quote. SynFutures@v1 requires that the deviation between this price and the current index price shall not exceed the ratio defined by the global parameter maxInitialDailyBasis.
- leverage is the specified leverage multiple, to be explained in detail later.
- deadline limits the transaction execution time to be no later than this specified time defined by the elapsed seconds of UTC time, same as what the block timestamp of Ethereum uses. The last parameter of all non-query operations in SynFutures@v1 is the deadline

In the design of sAMM, there are only LONG positions in AMM accounts, which are fully collateralized to ensure that AMM accounts are always safe. As mentioned earlier, half of the liquidity added by LP is used as Quote, and the other half to synthesize AMM's LONG positions. In order to hedge AMM's LONG positions, LP will passively be assigned the same amount of SHORT positions after providing liquidity. To ensure the safety of the LP account' s SHORT position, the margin deposited by the LP in the contract should at least meet the initial margin requirement of the synthetic position. The leverage parameter is used to specify the leverage of the SHORT hedge position. Assuming that LP synthesizes size LONG positions in AMM through this operation, the relationship between each parameter and size is:

$$\text{wadAmount} = \text{initPrice} \times \text{size} \times 2 + \text{initPrice} \times \text{size} \;/\; \text{leverage}$$

$$\text{size} = \text{wadAmount} \;/\; (2 \times \text{initPrice} + \text{initPrice} \;/\; \text{leverage})$$

In order to prevent the initPrice specified by LP from being unreasonable, SynFutures@v1 specifies the global parameter maxInitialDailyBasis to restrict the absolute value of the price difference between initPrice and the Spot index price given by Oracle not to exceed days × maxInitialDailyBasis, where days is the duration of the futures contract. After the depositAndInitPool() operation is completed, the LP will obtain the corresponding LP Token. The name of the LP Token follows the BASE-QUOTE-EXPIRY-ORACLETYPE model. For example, the LP Token representing BTC/USD contract expires on December 25, 2020, with Chainlink as the oracle will be assigned the symbol BTC-USD-20201225-LINK. In addition, depositAndInitPool() will also initialize the state of MarkPriceState. MarkPriceState is used to assist in the calculation of mark price during the entire life cycle of the futures contract, which will be introduced later.

After sAMM is initialized, LP can continue to provide liquidity into AMM through two-step operations deposit() and addLiquidity(). Of course, it can also complete the two-step operations at one time through the depositAndAddLiquidity() method provided by SynFutures@v1, whose input parameters are basically the same as depositAndInitPool(), except that the parameter used to specify the initial price is not required. This is because after sAMM is initialized, subsequent liquidity provision must follow the fair price quoted by sAMM.

```
// deposit margin and add liquidity to pool, margin token amount and leverage in 10^18
function depositAndAddLiquidity(
    uint wadAmount, uint leverage, uint deadline
) public payable onlyTrading returns (bool, uint)
```

The FairPrice quoted by sAMM would be x / y, where x represents the available margin in the AMM account, and y is the LONG position currently held by AMM. The depositAndAddLiquidity() method can also adjust the amount of funds provided into AMM and left in the LP account in wadAmount through the leverage parameter. Note that the depositAndAddLiquidity() operation is only allowed in the TRADING state.

As mentioned earlier, LP can also complete the above process in two steps: deposit() and addLiquidity(), and addLiquidity() is also only allowed in the TRADING state. After providing liquidity into sAMM, in the TRADING and SETTLING states, LP can remove liquidity at the current price of sAMM through the removeLiquidity() operation and close the hedge position, and then withdraw the assets through withdraw(); and in the SETTLED state, LP can use settle() operation to remove liquidity according to the settlement price, close all positions in the account (including Long/Short positions opened as a trader) and withdraw funds at the same time. In the TRADING and SETTLING states, removeLiquidity() will reduce the liquidity in AMM. In order to prevent extremely low liquidity in AMM and excessive slippage in subsequent transactions, SynFutures@v1 requires a LONG position held by AMM after the liquidity is removed via removeLiquidity() shall not be lower than a certain proportion of the open interests of the futures contract. This ratio is specified in the global parameter minAmmOpenInterestRatio.

8.2 Buy/Long and Sell/Short

After LP creates a futures contract and provides liquidity, the trader can buy/long or sell/short against sAMM. SynFutures@v1 uses the trade() method to represent buy or sell operation, the specific transaction direction is distinguished according to the first Boolean parameter buy. The parameter size indicates the number of positions to be traded, and limitPrice is used to limit the trading slippage.

```
// trade with AMM
function trade(bool buy, uint size, uint limitPrice, uint deadline) public
    onlyTradingOrSettling returns (bool)
```

The transaction price is calculated by AMM based on the available margin x, LONG position y in AMM, and transaction size when the transaction occurs. It is worth noting that during SETTLING phase, new positions are not allowed to be opened, and the actual tradeable size of the trader is the minimum value of the number specified by the parameter and the number of current account positions.

- Buy/Long: price = x / (y - size), here limitPrice is the upper limit of the buy price, which means that limitPrice $\geq$ price
- Sell/Short: price = x / (y + size), here limitPrice is the lower limit of the sell price, which requires price $\geq$ limitPrice

It can be seen that the denominator of buy/long is (y - size), so size $<$ y is required. Buy/Long will reduce the number of AMM's LONG positions, so similar to removeLiquidity(), in buy/long, SynFutures@v1 also requires that after the transaction, the proportion of AMM's LONG positions shall not be lower than the open interest of the futures contract, or minAmmOpenInterestRatio, otherwise the transaction fails.

In order to prevent a certain contract from being too concentrated in the hands of limited users, and to mitigate systemic risks especially during liquidation, SynFutures@v1 checks the user's position v.s. open interest ratio when he or she opens new positions. If the ratio is too high (specified by the global parameter maxUserTradeOpenInterestRatio), the entire transaction will fail, that is, the user can only close but not increase positions at this time. It is worth mentioning that this check is only performed in the TRADING state, and there will not be any new position in the trader account in the SETTLING stage due to the previously explained checking.

Taking into account the nature of futures contracts operating with smart contracts, to protect user' s interests, SynFutures@v1 also introduces another constraint parameter maxPriceSlippageRatio, which is used to limit the maximum two-way price slippage in a block when executing transactions, or effectively introducing a price limit for all transactions in each block. This prevents price distortion and manipulation by a single large transaction. Therefore, in each trade operation, it will confirm with the latest state of AMM that the block price slippage limit has not been breached, otherwise the transaction will fail.

SynFutures@v1 charges a fixed proportion of fees for all trades based on the transaction amount, including two parts of fees, one is system reserve fees (for liquidation initiator, etc) and the other is transaction pool fees (for LPs). The fee ratios are specified by the global parameters poolReserveFeeRatio and poolFeeRatio respectively.

At the end of a trade operation, the contract will also ensure that the user account and the AMM account are safe, which means that if the user account has a new position, the account is required to be IMSafe at the current mark price:

$$\text{AccountBalance} + \text{UnrealizedPnl} \geq \text{Position} \times \text{MarkPrice} \times \text{IMR},$$

otherwise, the account is required to be MMSafe:

$$\text{AccountBalance} + \text{UnrealizedPnl} \geq \text{Position} \times \text{MarkPrice} \times \text{MMR}.$$

In addition, it is also required that the AMM account is also MMSafe: according to the design, AMM is always safe and this check is just for prudent purposes, which does not consume a lot of gas fee. The calculation method of index prices will be explained in detail in the next section.

9 Price

There are three types of prices in SynFutures@v1: Index Price from Oracle, Fair Price from AMM, and Mark Price maintained and continuously updated through AMM.

In terms of Index Prices, for Uniswap as Oracle, SynFutures@v1 directly reads the quantity numbers of two assets in the underlying trading pair in Uniswap, and divides them to get the index price. In order to minimize the price fluctuation caused by price manipulation, the global parameter maxSpotIndex-ChangePerSecondRatio was introduced in SynFutures@v1 to limit the index prices from Uniswap trading

pairs: If the index price input exceeds the allowable range of this parameter, the max allowable value is used as the index price. For Chainlink as Oralce, SynFutures@v1 has no special handling, but treats Chainlink's USD price quotation as the quotation for USDC.

When initializing AMM (adding liquidity to AMM for the first time), the initial price specified should be within a reasonable range of the current index price. Subsequent operations such as adding/removing liquidity, buy/long and sell/ short, etc. are executed according to the Fair Price , representing market price provided by AMM, but when determining whether the account is safe and liquidation could be triggered, the contract would refer to Mark Price.

As mentioned earlier, depositAndInitPool() will initialize the state of MarkPriceState, and MarkPriceState is used to assist in the calculation of Mark Price during the entire life cycle of the futures contract.

```
struct MarkPriceState {
    uint32 lastMarkTime; // block time of the last update
    uint112 lastIndexPrice; // oracle index price
    int112 lastEmaBasis; // EMA of futures price over index price
    uint32 accIndexStartTime; // for index price TWAP calculation
    uint224 accIndexPrice; // for index price TWAP calculation
}
```

In the TRADING and SETTLING phases, all operations except deposit(), including withdraw(), update(), trade(), addLiquidity(), removeLiquidity(), liquidate(), liquidateByAmm() will first try to update the status of the futures contract (TRADING → SETTLILNG → SETTLED) based on the current time and contract expiration time, and also the MarkPriceState according to the current index price and current time. In particular, when switching from the SETTLLING state to the SETTELD state, the settlement price of the futures contract will be calculated. While after entering the SETTLED state, there is no need to update the futures contract state and the MarkPriceState.

```
// calculate mark price from given mark price state
function _markPriceWithState(Types.MarkPriceState memory state) internal view returns (uint) {
    if (status == Types.Status.TRADING) {
        int index = uint(state.lastIndexPrice).toInt256();
        int p = index.add((state.lastEmaBasis));
        return p.max(0).toUint256();
    } else if (status == Types.Status.SETTLING) {
        return _twapAfterSettling(state);
    } else { // SETTLED or EMERGENCY
        return settlementPrice; // just return settlementPrice while settled
    }
}
```

- In the TRADING state, only the fields starting with last in MarkPriceState are updated and the mark price is calculated according to EMA (exponential moving average) with the current MarkPriceState and the information provided by Oracle.
- In the SETTLING state, fields accIndexStartTime and accIndexPrice in MarkPriceState will be updated, and the mark price will be calculated according to TWAP (time weighted average price) with the current MarkPriceState and the information provided by Oracle.
- In the SETTLED state, there is no need to update the MarkPriceState, and the mark price is the settlement price.

9.1 MarkPriceState update in EMA mode in TRADING state

In the TRADING state, The mark price in SynFutures@v1 is defined as the spot index price plus the mark basis. The spot index reflects real-time changes in the spot market, and mark basis uses exponential moving average (EMA) to record the difference between futures and spot. In the TRADING state, the mark price MarkPrice_T at time T is calculated as followed:

$$\text{Basis}_T = \text{FairPrice}_T - \text{IndexPrice}_T$$

$$\text{MarkBasis}_T = \alpha \cdot \text{Basis}_T + (1-\alpha) \cdot \text{MarkBasis}_{T-1}, \text{ where } \alpha = 1 - e^{-\Delta T/\tau}$$

$$\text{MarkPrice}_T = \text{IndexPrice}_T + \text{MarkBasis}_T$$

Among them, τ represents the global parameter emaTimeConstant of SynFutures@v1 in seconds, and the maximum value is 86400 seconds, which is 24 hours; ΔT represents the number of seconds that have elapsed since the last update of MarkPriceState to the current moment. The last update time is recorded in the lastMarkTime field of MarkPriceState, and the index price of the last update is recorded in the lastIndexPrice field.

It is worth mentioning that for unusually inactive futures contracts, there may be cases where the value of $\Delta T/\tau$ is large, such as exceeding 4 or even exceeding 5 and would not be friendly for the approximate algorithm of calculating $e^{-\Delta T/\tau}$. On the other hand, $e^{-4.6} \approx 0.0100518357446336$, which means in the calculation of MarkBasis_T, MarkBasis_{T-1} has only 1% weighting, and the weight will be further reduced as the absolute value of the index increases, resulting in MarkBasis_{T-1} having almost no effect on MarkBasis_T. Thus, when the absolute value of the index of $e^{-\Delta T/\tau}$ exceeds 4.6, it is considered that MarkBasis_{T-1} holds 1% of the weight in SynFutures@v1.

9.2 MarkPriceState update in TWAP mode in SETTLING state

In the SETTLING state, the calculation of the mark price in SynFutures@v1 adopts the TWAP, or time weighted average price method, and the accIndexStartTime and accIndexPrice are continuously updated to assist the calculation of the mark price. The update method varies according to the types of Oracle.

Uniswap type Oracle will directly read the quantity balance of the target trading pair in the TRADING state and use the quotation as the index price, with additional control parameters. In the SETTLING state, since SynFutures@v1 switches to TWAP to calculate the mark price, and the Uniswap V2 trading pair itself has the accumulated price over time, the calculation method will be as follows: firstly the time when entering the SETTLING state is recorded in the accIndexStartTime field of MarkPriceState, and the accumulated price in the appropriate direction in Uniswap at this time is read and stored in the accIndexPrice field of MarkPriceState. Subsequent SETTLING phase does not update the value of this field. Every time the mark price needs to be calculated, the accumulated price of Uniswap will be read again, and TWAP will be calculated as the mark price based on accIndexPrice and accIndexStartTime.

Chainlink type Oralce directly reads its feed price as index price in the TRADING and SETTLING phases. But when entering the SETTLING phase, SynFutures@v1 will also record the start time in the accIndexStartTime field of MarkPriceState, and record the index price accumulated in seconds in the

accIndexPrice field of MarkPriceState. Every time the mark price needs to be calculated, the TWAP since SETTLING is calculated according to accIndexStartTime and accIndexPrice.

10 Liquidation

In SynFutures@v1, if an account is no longer safe according to the current mark price (AccountBalance + UnrealizedPnl $<$ Position $\times$ MarkPrice $\times$ MMR), any user of the current futures contract can initiate a liquidation operation for the account. SynFutures@v1 supports two types of liquidate operations.

liquidate(): The liquidation initiator takes over all positions of the liquidated account at the current mark price. After all positions of the liquidated account are closed, a penalty (deducted from the current account balance) is paid to the insurance fund according to the total value of the liquidation, the ratio is specified by the global parameter insurancePremiumRatio and the liquidation initiator gets all leftover balances of the liquidated account. However, if the balance of the liquidated account after closing the position is not enough to pay the penalty, the liquidation initiator may not be able to obtain additional rewards. To solve this problem, a certain proportion of rewards will be given to the liquidation initiator from the insurance fund in this case according to the total amount of liquidation, which is specified by the global parameter bankruptcyLiquidatorRewardRatio. By now, it means that the balance of the liquidated account cannot cover all losses and the insurance fund is the first resort to cover. If the balance in the insurance fund is not enough to fill the shortfall, the remaining part will be borne by the counter side of the liquidated account who holds an opposition direction position proportionally, that is, the social loss of futures contracts will be updated. Meanwhile, the liquidation initiator is required to keep its account safe after taking over all positions: if a new position is opened during the process, the requirement is IMSafe; if there is no new position opened in the liquidation initiator' s account, the requirement is MMSafe.

liquidateByAmm(): To lower the liquidity requirement, liquidation initiators could utilize this method to force the liquidated account to trade with AMM with the same price logic as the trade() method. Note that if the position that needs to be liquidated causes excessive price fluctuations in AMM after being traded to AMM, the liquidation process will fail. That is, all the constraints in trade method still apply here. Similarly, after forcing the liquidated account to close a certain amount of positions, the account balance of the liquidated account may become negative. Same as above, insurance fund and social loss will become the resort. liquidateByAmm() will give the liquidation initiator a fixed reward, same as the operation to update contract status. It is worth mentioning that since AMM has always maintained liquidity in the system, partial liquidation becomes feasible under this operation: unlike liquidate() method that takes over all positions of the liquidated account, with liquidateByAmm(), the account will only be partially liquidated to meets the initial margin requirement.

11 Market Restriction and User Protection

The following mechanisms are mainly intended to protect the entire ecosystem and users from being attacked by abnormal market volatility, including manipulation of Oracle spot index prices through flash loans etc. Restriction parameters can be adjusted through GlobalConfig. Most protection mechanisms are only effective within one single block, and normal users will not perform multiple operations within the same block and thus will not be affected.

maxPriceSlippageRatio: maximum price deviation in a single block for either direction from the fair price at the start of the current block. This serves as a limit of price slippage for the AMM and protects the system from attacks distorting the market within the same block. A trade would be reverted if it results in a price breaching this limit of this block.

maxInitialDailyBasis: maximum deviation of initial futures price to spot index per day to limit the initial price for AMM in a reasonable range, e.g. $|\text{initPrice} - \text{indexPrice}| < \text{indexPrice} \times \text{days} \times \text{maxInitialDailyBasis}$.

maxUserTradeOpenInterestRatio: maximum open interest ratio of the entire market for a single user (address) to prevent concentration of risk in a single account. When a user's account has a higher open interest ratio than this limit, the user can only execute trades to reduce but not increase position. This limit does not apply to the action of LP adding liquidity to the AMM. But if an LP's trading position breaches the limit after adding liquidity to the AMM, they cannot increase their position further.

minAmmOpenInterestRatio: minimum open interest ratio of the entire market for the AMM to prevent a drain of liquidity. The AMM needs to maintain a certain level of inventory to prevent large slippages as every user can only trade with the AMM. This limit applies to both users buying from the AMM and LPs removing liquidity.

maxSpotIndexChangePerSecondRatio: maximum spot index change that can be accepted since the last update, measuring in seconds. As mark price is updated at most once per block, this serves as a limit of the mark price change per block and protects the system from attacks distorting the underlying oracle in a short period of time.

Besides the above restrictions, reward is offered to the system maintainer to further protect user interest. bankruptcyLiquidatorRewardRatio: represents the reward that users can get from the system when liquidating a bankrupt account. The insurance fund pays this reward to attract users to liquidate accounts that have gone bankrupt.