Tech description

Compound protocol

Consists of two parts: - Main lend&borrow logic. - Interaction with many users and tokens. Algorithm for finding risk weights, interest rate etc.

Borrowing

CToken

It is fungible token, so in addition to the default functions(approve, transfer, burn), we will add the following:

```
// Return the underlying balance of the `owner`
fn balance_underlying(owner: ActorId) -> u128;

// Calculates interest accrued from the last checkpointed block up to the current block and
fn accrue_interest() -> u128;

// `user_address` supplies assets into the market and receives cTokens in exchange
// `amount` - The amount of the underlying asset to supply
fn mint_action(user_address: ActorId, amount: u128);

// Consists of `accrue_interest` and `mint_action` with `msg::source()` as argument
fn mint(amount: u128);

// `user_address` redeems cTokens in exchange for the underlying asset
// `amount` - The number of cTokens to redeem
 fn redeem_action(user_address: ActorId, amount: u128);

// Consists of `accrue_interest` and `redeem_action` with `msg::source()` as argument
fn redeem(amount: u128);

// `user_address` borrow assets from the protocol to their own address
// `amount` - The amount of the underlying asset to borrow
fn borrow_action(user_address: ActorId, amount: u128);

// Consists of `accrue_interest` and `borrow_action` with `msg::source()` as argument
fn borrow(amount: u128);

// Borrows are repaid by another user (possibly the borrower) and return the actual repaymen
// `payer` - The account paying off the borrow
// `borrower` - The account with the debt being payed off
// `amount` - The amount of underlying tokens being returned
fn repay_borrow_action(payer: ActorId, borrower: ActorId, amount: u128) -> u128;
```

```
// Consists of `accrue_interest` and `repay_borrow_action` with `msg::source()` as argument
fn repay_borrow(amount: u128);

// `liquidator` liquidates `borrower` collateral. The collateral seized is transferred to th
// `liquidator` - The address repaying the borrow and seizing collateral
// `borrower` - The borrower of this cToken to be liquidated
// `amount` - The amount of the underlying borrowed asset to repay
// `market` - The market in which to seize collateral from the borrower
fn liquidate_borrow_action(liquidator: ActorId, borrower: ActorId, amount: u128, market: ...

// Consists of `accrue_interest`, `accrue_interest` for `market` and `liquidate_borrow_actio
fn liquidate_borrow(borrower: ActorId, amount: u128, market: ...);

// Transfers collateral tokens (this market) to the `liquidator`
// `seizer_token` - The contract seizing the collateral (i.e. borrowed cToken)
// `liquidator` - The account receiving seized collateral
// `borrower` - The account having collateral seized
// `amount` - The number of cTokens to seize
fn seize(seizer_token: ActorId, liquidator: ActorId, borrower: ActorId, amount: u128);
```

DeFi

Some functions (in progress):

```
- fn enter_market(address: ActorId);
- fn exit_market(address: ActorId);
- ...
```