B_weth.mch

```

MACHINE
    B_weth
SEES Solidity_Types
INCLUDES
    Platform, account, allowance
ABSTRACT_CONSTANTS
    threshold
PROPERTIES
    threshold  = 2
VARIABLES
    manager, depositors, donated
INVARIANT
   depositors <:  ADDRESS &
   manager : USERS &
   donated : BOOL &
   balanceOf(THIS) >=
   (SIGMA(ct).(ct : dom(accountOf)| accountOf(ct))) &
   card(depositors)<= threshold

INITIALISATION manager := init_msg_sender  || depositors := {} || donated := FALSE
OPERATIONS
    deposit(msg_sender, msg_value) =
    PRE
        msg_sender : USERS & msg_value : NAT1
    THEN
        IF balanceOf(msg_sender) - msg_value : NAT & accountOf(msg_sender) + msg_value
: NAT &
        balanceOf(THIS) + msg_value: NAT THEN
            transfer(msg_sender, THIS, msg_value)||
            set_accountOf_abstract({msg_sender
                |-> accountOf(msg_sender) + msg_value}) ||
            IF (accountOf(msg_sender) + msg_value >= threshold )
            & msg_sender /: depositors  & card(depositors) < threshold
            THEN
                depositors := depositors \/ {msg_sender}
            END
        END
    END
    ;
    withdraw(msg_sender, amount) =
    PRE
        msg_sender : USERS & amount: NAT1
    THEN
        IF  accountOf(msg_sender) >= amount & balanceOf(msg_sender) + amount: NAT &
        balanceOf(THIS) - amount  : NAT THEN
            transfer(THIS , msg_sender, amount)||
            set_accountOf_abstract({msg_sender |-> accountOf(msg_sender) -  amount})
        END
    END
    ;
    transferTo(msg_sender, dst, amount) =
    PRE
        msg_sender : USERS & dst: USERS & amount: NAT1
    THEN
        IF accountOf(msg_sender) > amount &
```

```
            // accountOf(msg_sender) - amount : NAT &  //pas utile
             msg_sender /= dst &
             accountOf(dst) + amount : NAT
         THEN
             set_accountOf_abstract({msg_sender |-> accountOf(msg_sender)
             - amount, dst |-> accountOf(dst) + amount })
         END
     END
     ;
     approve(msg_sender, dst, amount)=
     PRE
         msg_sender : USERS & dst : USERS & amount : NAT1
     THEN
         IF dst /= msg_sender THEN
             set_allowanceOf_abstract(msg_sender, { dst  |->amount})
         END
     END
     ;
     transferFrom(msg_sender, sender, recipient, amount ) =
     PRE
         msg_sender : USERS & sender : USERS &
         recipient : USERS & amount : NAT1
     THEN
         IF  sender /= recipient &
             allowanceOf(sender)(msg_sender) >= amount &
             accountOf(sender) >= amount &
             accountOf(recipient) + amount : NAT &
             allowanceOf(sender)(msg_sender) - amount : NAT
         THEN
             set_accountOf_abstract({recipient |->
               accountOf(recipient) + amount, sender
                 |-> accountOf(sender) - amount})
             ||
             set_allowanceOf_abstract(sender, {msg_sender
                 |-> allowanceOf(sender)(msg_sender) - amount})
         END
     END
     ;
     rewardTopDepositors(msg_sender, msg_value) =
     PRE
         msg_sender : USERS & msg_value : NAT
     THEN
         IF  msg_value = threshold &
             msg_sender = manager &
             card(depositors) = threshold &
             donated = FALSE &
             balanceOf(THIS) + msg_value : NAT &
             balanceOf(manager) - msg_value : NAT &
             !xx.(xx : depositors => accountOf(xx) + 1 : NAT)
         THEN
             transfer(manager, THIS, msg_value) ||
             set_accountOf_abstract(%xx. (xx : depositors | accountOf(xx) + 1)) ||
             donated := TRUE
         END
      END
END
```

B_weth_i.imp

```
/* B_weth_i
* Author: ASUS
* Creation date: 8/31/2023
*/

IMPLEMENTATION B_weth_i
REFINES B_weth
SEES Solidity_Types
IMPORTS Platform, account, allowance, depositedOver100
CONCRETE_CONSTANTS
    threshold_i
PROPERTIES
    threshold_i : NAT & threshold_i = threshold
VALUES
    threshold_i = 2
CONCRETE_VARIABLES manager_i, depositors_i, index, donated_i
INVARIANT
    index : NAT & index>=0 & donated_i : BOOL &
    depositors_i : 0..threshold_i --> ADDRESS &
    manager_i : USERS & manager_i = manager & donated_i = donated &
    index = card(depositors) &
    depositors_i[0..index-1] = depositors &
    (0..index-1) <| depositors_i : 0..index-1 >-> depositors &
    depositedOver_100~[{TRUE}] = depositors

INITIALISATION
    index := 0;
    depositors_i := (0..threshold_i) * {addr_0} ;
    manager_i := init_msg_sender;
    donated_i := FALSE

OPERATIONS
    deposit(msg_sender, msg_value) =
    BEGIN
        VAR senderBalance, senderAccount, thisBalance IN
            senderAccount <-- get_accountOf(msg_sender);
            senderBalance <-- get_balanceOf(msg_sender);
            thisBalance <-- get_balanceOf(THIS);

            IF thisBalance + msg_value <= MAXINT & senderBalance - msg_value >=0 &
                senderAccount + msg_value <= MAXINT
            THEN
                set_accountOf(msg_sender, senderAccount + msg_value);
                transfer(msg_sender, THIS, msg_value);
                VAR distinct IN
                    distinct <-- get_depositedOver_100(msg_sender);
                    IF senderAccount + msg_value >= threshold_i &  distinct = FALSE &
index < threshold_i
                    THEN
                        depositors_i(index) := msg_sender;
                        set_depositedOver_100(msg_sender, TRUE);
                        index := index +1
                    END
                END
            END
        END
    END
    ;

    withdraw(msg_sender, amount) =
    BEGIN
        VAR senderAccount, senderBalance, thisBalance IN
            senderAccount <-- get_accountOf(msg_sender);
            senderBalance <-- get_balanceOf(msg_sender);
            thisBalance <--get_balanceOf(THIS);
            IF senderAccount >= amount & senderBalance + amount <= MAXINT &
thisBalance>=amount
            THEN
                transfer(THIS , msg_sender, amount);
                set_accountOf(msg_sender, senderAccount -  amount)
            END
        END
    END
    ;

    transferTo(msg_sender, dst, amount) =
    BEGIN
        VAR senderBalance, receiverBalance IN
            senderBalance <-- get_accountOf(msg_sender);
            receiverBalance <-- get_accountOf(dst);
            IF senderBalance > amount & receiverBalance + amount <= MAXINT & msg_sender
/= dst
            THEN
                set_accountOf(msg_sender,  senderBalance - amount);
                set_accountOf(dst, receiverBalance + amount)
            END
        END
    END
    ;

    approve(msg_sender, dst, amount)=
    BEGIN
        IF msg_sender /= dst THEN
            set_allowanceOf(msg_sender, dst, amount)
        END
    END
    ;
    transferFrom(msg_sender, sender, recipient, amount) =
    BEGIN
        VAR senderBalance, recipientBalance, allowance IN
            senderBalance <-- get_accountOf(sender);
            recipientBalance <-- get_accountOf(recipient);
            allowance <-- get_allowanceOf(sender, msg_sender);
            IF sender /= recipient & allowance >= amount & senderBalance >= amount &
                recipientBalance + amount <= MAXINT
            THEN
                set_accountOf(sender, senderBalance - amount);
                set_accountOf(recipient, recipientBalance + amount);
                set_allowanceOf(sender, msg_sender, allowance - amount)
            END
        END
    END
    ;
    rewardTopDepositors(msg_sender, msg_value) =
    BEGIN
```

```
        VAR thisBalance, managerBalance IN
           thisBalance <-- get_balanceOf(THIS);
            managerBalance <-- get_balanceOf(manager_i);
            IF msg_value = threshold_i &
                msg_sender = manager_i &
                index = threshold_i &
                donated_i = FALSE &
                thisBalance + msg_value <= MAXINT &
                managerBalance - msg_value >= 0
            THEN
                //* jj : NAT;
                //* safe : BOOL;
                VAR jj, safe IN
                    jj := 0;
                    safe := TRUE;
                    WHILE jj < index & safe = TRUE DO
                        VAR depositorBalance IN
                            depositorBalance <-- get_accountOf(depositors_i(jj));
                            safe := bool(depositorBalance + 1 <= MAXINT);
                            jj := jj+ 1
                        END
                    INVARIANT 0<=index & jj<=index   &  jj>=0 &
                        safe = bool(!xx.(xx : ran((0..jj-1) <| depositors_i) => accountOf(xx) + 1 : NAT)) &
                        donated_i= FALSE  &
                        !xx.(xx : ran((0..jj-2) <| depositors_i) => accountOf(xx) + 1 : NAT)
                    VARIANT index - jj
                    END;

                    IF (safe=TRUE) THEN
                        transfer(msg_sender, THIS, msg_value);
                        donated_i := TRUE;
                        //* ii : NAT;
                        VAR ii, depositorBalance IN
                            ii := 0;
                            WHILE ii < index DO
                                depositorBalance <-- get_accountOf(depositors_i(ii));
                                set_accountOf(depositors_i (ii), depositorBalance + 1);
                                ii := ii+ 1
                            INVARIANT ii=threshold_i or ii: dom(depositors_i) &
                                accountOf =
                                accountOf$0<+(%xx. (xx : depositors_i[0..(ii-1)] | accountOf$0(xx) + 1)) &
                                threshold_i = threshold &
                                donated_i = TRUE & safe= TRUE &
                                depositors_i[0..(ii-1)]<: depositors &
                                jj=index &
                                !xx.(xx : ran((ii+1..index-1) <| depositors_i) => accountOf(xx) + 1 : NAT)
                            VARIANT index - ii
                            END
                        END
                    END
                END
            END
        END
    END
END
```

Platform.mch

```

MACHINE
    Platform
SEES
    Solidity_Types
ABSTRACT_VARIABLES
    balanceOf
INVARIANT
    balanceOf : ADDRESS --> NAT
INITIALISATION
   balanceOf :: ADDRESS --> NAT
OPERATIONS

    // Used in animating model in proB.
    addRandomAmountToBalance =
    BEGIN
        ANY amount_, xx WHERE xx : USERS & amount_ : 1..3 & balanceOf(xx) + amount_ :
NAT THEN
                balanceOf(xx) := balanceOf(xx) + amount_
        END
    END
    ;
    transfer ( from , to , amount ) =
     PRE
        from : ADDRESS &
        to : ADDRESS &
        to /= from &
        amount : NAT &
        ( balanceOf ( to ) + amount ) : NAT &
        ( balanceOf ( from ) - amount ) : NAT
    THEN
        balanceOf := balanceOf <+ { from |-> ( balanceOf ( from ) - amount ) , to |-> (
balanceOf ( to ) + amount ) }
    END
    ;
    transfer_abstract(updates) =
     PRE
       updates : ADDRESS +-> NAT
    THEN
        balanceOf := balanceOf <+ updates
    END
    ;
    ret <-- get_balanceOf ( adr ) =
    PRE
        adr : ADDRESS
    THEN
    ret := balanceOf ( adr )
    END
    ;
    // Test if model is vulnerable to ForceFeeding
    ForceFeeding (amount) =
    PRE amount : NAT & balanceOf(THIS) + amount : NAT
    THEN
        balanceOf := balanceOf <+ {THIS |-> ( balanceOf ( THIS ) + amount )}
    END
    ;
    transfer_(balanceUpdates) =
    PRE
        balanceUpdates : ADDRESS +-> NAT
    THEN
        balanceOf := balanceOf <+ balanceUpdates
    END
END
```

Solidity_Types.mch

```

MACHINE
    Solidity_Types
SETS
    ADDRESS = {addr_0, THIS, addr_1, addr_2, addr_3}; BYTES
CONSTANTS
    init_msg_sender, init_msg_value, USERS, init_block_timestamp
PROPERTIES
    USERS = ADDRESS - {THIS, addr_0} & init_msg_sender : USERS & init_msg_value : NAT &
init_block_timestamp : NAT


END
```

account.mch

```

MACHINE
    account
SEES
    Solidity_Types
VARIABLES
    accountOf
INVARIANT
    accountOf : ADDRESS --> NAT
INITIALISATION
    accountOf := ADDRESS * {0}

OPERATIONS

    ret <-- get_accountOf(key) =
    PRE
        key : ADDRESS
    THEN
        ret := accountOf(key)
    END
    ;
    set_accountOf_abstract(updates) =
    PRE
        updates : ADDRESS +-> NAT

    THEN
        accountOf := accountOf <+ updates
    END
    ;
    set_accountOf(key, value) =
    PRE
        key : ADDRESS & value : NAT
    THEN
        accountOf(key) := value
    END
    ;

    ret <-- get_account = ret := accountOf

END

```

allowance.mch

```

MACHINE
    allowance

SEES
    Solidity_Types

VARIABLES
    allowanceOf
INVARIANT
    allowanceOf : ADDRESS --> ( ADDRESS--> NAT )


INITIALISATION
    allowanceOf :: (ADDRESS --> ( ADDRESS --> NAT ))

OPERATIONS

    set_allowanceOf_abstract( key, updates ) =
    PRE
        key : ADDRESS & updates : ADDRESS +-> NAT
    THEN
        allowanceOf(key) := allowanceOf(key) <+ updates
    END
    ;


    ret <-- get_allowanceOf ( key1, key2 ) =
    PRE
        key1 : ADDRESS & key2 : ADDRESS
    THEN
        ret := allowanceOf ( key1 )( key2 )
    END
    ;

    set_allowanceOf(key1, key2, value) =
    PRE
        key1: ADDRESS &  key2 : ADDRESS & value: NAT
    THEN
        allowanceOf(key1)(key2) := value
    END


END
```

depositedOver100.mch

```

MACHINE
    depositedOver100
SEES
    Solidity_Types
VARIABLES
    depositedOver_100
INVARIANT
    depositedOver_100: ADDRESS --> BOOL
INITIALISATION
    depositedOver_100 := ADDRESS * {FALSE}

OPERATIONS

     set_depositedOver_100_abstract(updates) =
     PRE updates : ADDRESS +-> BOOL
     THEN
        depositedOver_100 := depositedOver_100 <+ updates
     END
     ;

     set_depositedOver_100(key, value) =
     PRE key : ADDRESS & value : BOOL
     THEN
        depositedOver_100(key) := value
     END
     ;

    ret <-- get_depositedOver_100(key) =
    PRE key : ADDRESS
    THEN
        ret := depositedOver_100(key)
    END
END

```