A THE ALGORITHMS FOR EXTRACTING DEFI SEMANTICS

Algorithm 1: Extracting Contract Ownership

Input: Contract C
Output: Owner *Owner* of C

```
Owner ← null;
foreach Instruction I ∈ C do
    if type(I) = REQ then
        O1 ← I.leftVal;
        O2 ← I.rightVal;
        if type(O1) = address ∧ O2 = msg.sender then
            Owner ← O1 ;                    // singular owner
        else if type(O2) = address ∧ O1 = msg.sender
         then
            Owner ← O2 ;                    // singular owner
        else if type(O1.base) = map {address ⇒ bool} ∧
         O1.index = msg.sender ∧ O2 = true then
            Owner ← O1.base ;               // owner group
        else if type(O2.base) = map {address ⇒ bool} ∧
         O2.index = msg.sender ∧ O1 = true then
            Owner ← O2.base ;               // owner group
return Owner
```

Algorithm 2: Extracting Contract Initializer

Input: Contract C
Output: Initialization Flag *IF* and Initializer *IR* of C

```
IF ← null;
IR ← null;
foreach Function F⃗ ∈ C do
    foreach Control Flow P ∈ F⃗ do
        tmpIF ← null;
        foreach Instruction I ∈ P do
            if type(I) = REQ then
                O1 ← I.leftVal;
                O2 ← I.rightVal;
                if type(O1) in {bool, int} ∧ O2 = 0 then
                    tmpIF ← O1;
                else if type(O2) ∈ {bool, int} ∧ O1 = 0
                 then
                    tmpIF ← O2;
            if tmpIF ≠ null ∧ type(I) = SSTORE ∧
             I.target = tmpIF ∧ I.source ≠ 0 then
                IF ← tmpIF;
                IR ← F⃗;
return {IF, IR}
```

Algorithm 3: Extracting ERC-20 Balances

Input: ERC-20 Contract C
Output: ERC-20 Balances Mapping B of C

```
Balances ← null;
foreach Function F⃗ ∈ C do
    if type(F⃗) =
     transferFrom(address,address,uint256) then
        sender ← F⃗.args[0];
        receiver ← F⃗.args[1];
        amount ← F⃗.args[2];
    else if type(F⃗) = transfer(address,uint256) then
        sender ← msg.sender;
        receiver ← F⃗.args[0];
        amount ← F⃗.args[1];
    else
        continue;
    addMap ← null;
    subMap ← null;
    foreach Control Flow P ∈ F⃗ do
        foreach Instruction I ∈ P do
            if type(I) = ADD ∧ type(I.target.base) =
             map {address ⇒ uint} ∧
             defSource(I.addValue) = amount ;
             /* balance[to] += amount (- fee)? */
            then
                addMap ← I.target.base;
            else if type(I) = SUB ∧ type(I.target.base) =
             map {address ⇒ uint} ∧ I.subValue =
             amount ;   /* balance[from] -= amount */
            then
                subMap ← I.target.base;
    if addMap ≠ null ∧ addMap = subMap then
        Balances ← addMap;
return Balances
```

Algorithm 4: Extracting ERC-721 Owners

Input: ERC-721 Contract C
Output: ERC-721 Owners Mapping *Owners* of C

Owners ← `null`;
foreach Function $\vec{\mathcal{F}} \in C$ **do**
  if `type`($\vec{\mathcal{F}}$) ≠ `transferFrom(address,address,uint256)` **then**
    continue;
  sender ← $\vec{\mathcal{F}}$.args[0];
  receiver ← $\vec{\mathcal{F}}$.args[1];
  tokenId ← $\vec{\mathcal{F}}$.args[2];
  checkMap ← `null`;
  assignMap ← `null`;
  foreach Control Flow $\mathcal{P} \in \vec{\mathcal{F}}$ **do**
    foreach Instruction $\mathcal{I} \in \mathcal{P}$ **do**
      if `type`($\mathcal{I}$) = `REQ` ; `/* require(owner[tokenId] == from) */` **then**
        O_1 ← $\mathcal{I}$.leftVal;
        O_2 ← $\mathcal{I}$.rightVal;
        if `type`(O_1.base) = `map {uint ⇒ address}` ∧ O_1.index = *tokenId* ∧ O_2 = *sender* **then**
          checkMap ← O_1.base;
        else if `type`(O_2.base) = `map {uint ⇒ address}` ∧ O_2.index = *tokenId* ∧ O_1 = *sender* **then**
          checkMap ← O_2.base;
      else if `type`($\mathcal{I}$) = `SSTORE` ∧ $\mathcal{I}$.target.index = *tokenId* ∧ $\mathcal{I}$.source = *receiver;* `/* owner[tokenId] = to */` **then**
        assignMap ← $\mathcal{I}$.target.base;
    if *checkMap* ≠ `null` ∧ *checkMap* = *assignMap* **then**
      Owners ← *checkMap*;
return *Owners*

Algorithm 5: Extracting Critical States in Reflect Token

Input: ERC-20 Contract C
Output: {*rOwned*, *tOwned*, *rTotal*, *tTotal*, *rate*} if C is Reflect Token else {`null`}

rOwned ← `null`;
tOwned ← `null`;
rTotal ← `null`;
tTotal ← `null`;
rate ← `null`;
foreach Function $\vec{\mathcal{F}} \in C$ **do**
  if `type`($\vec{\mathcal{F}}$) ≠ `balanceOf(address)` **then**
    continue;
  foreach Control Flow $\mathcal{P} \in \vec{\mathcal{F}}$ **do**
    returnInst ← $\mathcal{P}$.returnInst;
    if `type`(*returnInst*) = `SLOAD` ; `/* tOwned[user] */` **then**
      tOwned ← *returnInst*.base;
    else if `type`(*returnInst*) = `DIV` ; `/*` $\frac{\texttt{rOwned[user]}}{\texttt{rate}}$ `*/` **then**
      rOwned ← *returnInst*.dividend.base;
      if `type`(*returnInst*.divisor) = `DIV` ; `/* rate =` $\frac{\texttt{rSupply}}{\texttt{tSupply}}$ `*/` **then**
        rate ← *returnInst*.divisor;
        if `type(defSource (`*rate*.dividend`))` = `SLOAD` ; `/* defSource yields the root of the def-use tree. */` **then**
          rTotal ← `defSource (`*rate*.dividend`)`
        if `type(defSource (`*rate*.divisor`))` ∈ {`CONST`, `SLOAD`} **then**
          tTotal ← `defSource (`*rate*.divisor`)`
if *rate* = `null` ; `/* C is normal ERC-20 */` **then**
  rOwned ← `null` ; `// reset to null`
  tOwned ← `null`;
return {*rOwned*, *tOwned*, *rTotal*, *tTotal*, *rate*}