A THE ALGORITHMS FOR EXTRACTING DEFI SEMANTICS

Algorithm 1: Extracting Contract Ownership

```
Input: Contract $C$
Output: Owner $Owner$ of $C$
1  $Owner \leftarrow$ null;
2  foreach Instruction $\mathcal{I} \in C$ do
3      if type($\mathcal{I}$) = REQ then
4          $O_1 \leftarrow \mathcal{I}$.leftVal;
5          $O_2 \leftarrow \mathcal{I}$.rightVal;
6          if type($O_1$) = address $\wedge$ $O_2$ = msg.sender then
7              $Owner \leftarrow O_1$ ;                // singular owner
8          else if type($O_2$) = address $\wedge$ $O_1$ = msg.sender then
9              $Owner \leftarrow O_2$ ;                // singular owner
10         else if type($O_1$.base) = map{address $\Rightarrow$ bool} $\wedge$ $O_1$.index = msg.sender $\wedge$ $O_2$ = true then
11             $Owner \leftarrow O_1$.base ;           // owner group
12         else if type($O_2$.base) = map{address $\Rightarrow$ bool} $\wedge$ $O_2$.index = msg.sender $\wedge$ $O_1$ = true then
13             $Owner \leftarrow O_2$.base ;           // owner group
14 return $Owner$
```

Algorithm 2: Extracting Contract Initializer

```
Input: Contract $C$
Output: Initialization Flag $IF$ and Initializer $IR$ of $C$
1  $IF \leftarrow$ null;
2  $IR \leftarrow$ null;
3  foreach Function $\vec{\mathcal{F}} \in C$ do
4      foreach Control Flow $\mathcal{P} \in \vec{\mathcal{F}}$ do
5          $tmpIF \leftarrow$ null;
6          foreach Instruction $\mathcal{I} \in \mathcal{P}$ do
7              if type($\mathcal{I}$) = REQ then
8                  $O_1 \leftarrow \mathcal{I}$.leftVal;
9                  $O_2 \leftarrow \mathcal{I}$.rightVal;
10                 if type($O_1$) in {bool, int} $\wedge$ $O_2 = 0$ then
11                     $tmpIF \leftarrow O_1$;
12                 else if type($O_2$) $\in$ {bool, int} $\wedge$ $O_1 = 0$ then
13                     $tmpIF \leftarrow O_2$;
14             if $tmpIF \neq$ null $\wedge$ type($\mathcal{I}$) = SSTORE $\wedge$ $\mathcal{I}$.target = $tmpIF$ $\wedge$ $\mathcal{I}$.source $\neq 0$ then
15                 $IF \leftarrow tmpIF$;
16                 $IR \leftarrow \vec{\mathcal{F}}$;
17 return {$IF$, $IR$}
```

Algorithm 3: Extracting ERC-20 Balances

```
Input: ERC-20 Contract $C$
Output: ERC-20 Balances Mapping $B$ of $C$
1  $Balances \leftarrow$ null;
2  foreach Function $\vec{\mathcal{F}} \in C$ do
3      if type($\vec{\mathcal{F}}$) = transferFrom(address,address,uint256) then
4          $sender \leftarrow \vec{\mathcal{F}}$.args[0];
5          $receiver \leftarrow \vec{\mathcal{F}}$.args[1];
6          $amount \leftarrow \vec{\mathcal{F}}$.args[2];
7      else if type($\vec{\mathcal{F}}$) = transfer(address,uint256) then
8          $sender \leftarrow$ msg.sender;
9          $receiver \leftarrow \vec{\mathcal{F}}$.args[0];
10         $amount \leftarrow \vec{\mathcal{F}}$.args[1];
11     else
12         continue;
13     $addMap \leftarrow$ null;
14     $subMap \leftarrow$ null;
15     foreach Control Flow $\mathcal{P} \in \vec{\mathcal{F}}$ do
16         foreach Instruction $\mathcal{I} \in \mathcal{P}$ do
17             if type($\mathcal{I}$) = ADD $\wedge$ type($\mathcal{I}$.target.base) = map{address $\Rightarrow$ uint} $\wedge$ defSource($\mathcal{I}$.addValue) = $amount$ ;
               /* balance[to] += amount (- fee)? */
18             then
19                 $addMap \leftarrow \mathcal{I}$.target.base;
20             else if type($\mathcal{I}$) = SUB $\wedge$ type($\mathcal{I}$.target.base) = map{address $\Rightarrow$ uint} $\wedge$ $\mathcal{I}$.subValue = $amount$ ;   /* balance[from] -= amount */
21             then
22                 $subMap \leftarrow \mathcal{I}$.target.base;
23     if $addMap \neq$ null $\wedge$ $addMap = subMap$ then
24         $Balances \leftarrow addMap$;
25 return $Balances$
```

Algorithm 4: Extracting ERC-721 Owners

```
Input: ERC-721 Contract $C$
Output: ERC-721 Owners Mapping $Owners$ of $C$
1  $Owners \leftarrow$ null;
2  foreach Function $\vec{\mathcal{F}} \in C$ do
3      if type($\vec{\mathcal{F}}$) $\neq$ transferFrom(address,address,uint256) then
4          continue;
5      $sender \leftarrow \vec{\mathcal{F}}$.args[0];
6      $receiver \leftarrow \vec{\mathcal{F}}$.args[1];
7      $tokenId \leftarrow \vec{\mathcal{F}}$.args[2];
8      $checkMap \leftarrow$ null;
9      $assignMap \leftarrow$ null;
10     foreach Control Flow $\mathcal{P} \in \vec{\mathcal{F}}$ do
11         foreach Instruction $\mathcal{I} \in \mathcal{P}$ do
12             if type($\mathcal{I}$) = REQ ;
                /* require(owner[tokenId] == from) */
13             then
14                 $O_1 \leftarrow \mathcal{I}$.leftVal;
15                 $O_2 \leftarrow \mathcal{I}$.rightVal;
16                 if type($O_1$.base) = map {uint $\Rightarrow$ address} $\wedge$ $O_1$.index = $tokenId$ $\wedge$ $O_2$ = $sender$ then
17                     $checkMap \leftarrow O_1$.base;
18                 else if type($O_2$.base) = map {uint $\Rightarrow$ address} $\wedge$ $O_2$.index = $tokenId$ $\wedge$ $O_1$ = $sender$ then
19                     $checkMap \leftarrow O_2$.base;
20             else if type($\mathcal{I}$) = SSTORE $\wedge$ $\mathcal{I}$.target.index = $tokenId$ $\wedge$ $\mathcal{I}$.source = $receiver$;
                /* owner[tokenId] = to */
21             then
22                 $assignMap \leftarrow \mathcal{I}$.target.base;
23         if $checkMap \neq$ null $\wedge$ $checkMap = assignMap$ then
24             $Owners \leftarrow checkMap$;
25 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}
1  $rOwned \leftarrow$ null;
2  $tOwned \leftarrow$ null;
3  $rTotal \leftarrow$ null;
4  $tTotal \leftarrow$ null;
5  $rate \leftarrow$ null;
6  foreach Function $\vec{\mathcal{F}} \in C$ do
7      if type($\vec{\mathcal{F}}$) $\neq$ balanceOf(address) then
8          continue;
9      foreach Control Flow $\mathcal{P} \in \vec{\mathcal{F}}$ do
10         $returnInst \leftarrow \mathcal{P}$.returnInst;
11         if type($returnInst$) = SLOAD ; /* tOwned[user] */
12         then
13             $tOwned \leftarrow returnInst$.base;
14         else if type($returnInst$) = DIV ; /* $\frac{\texttt{rOwned[user]}}{\texttt{rate}}$ */
15         then
16             $rOwned \leftarrow returnInst$.dividend.base;
17             if type($returnInst$.divisor) = DIV ;
                /* rate = $\frac{\texttt{rSupply}}{\texttt{tSupply}}$ */
18             then
19                 $rate \leftarrow returnInst$.divisor;
20                 if type(defSource ($rate$.dividend)) = SLOAD ;
                    /* defSource yields the root of the def-use tree. */
21                 then
22                     $rTotal \leftarrow$ defSource ($rate$.dividend)
23                 if type(defSource ($rate$.divisor)) $\in$ {CONST, SLOAD} then
24                     $tTotal \leftarrow$ defSource ($rate$.divisor)
25 if $rate$ = null ;                    /* $C$ is normal ERC-20 */
26 then
27     $rOwned \leftarrow$ null ;                 // reset to null
28     $tOwned \leftarrow$ null;
29 return {$rOwned$, $tOwned$, $rTotal$, $tTotal$, $rate$}
```