ETHER ANALYSIS OF SOLIDITY FUNCTIONS

COSIMO LANEVE   CLAUDIO SACERDOTI COEN

ORAL_COMMUNICATION@DLT2020

THE OBJECTIVE

analize assets movements of Solidity functions

evaluate the i/o beaviour of functions wrt contract balances

THE TECHNIQUE

1. extract an abstract description out of programs

    1.1. the descriptions – behavioural type – are extracted by means of a type system

    1.2. the behavioural types are cost equations – recurrence relations constrained by formulas – that compute ether movements of Solidity functions

2. use existing tools to solve cost equations about ether movements of functions

we have applied the same technique to

* compute overapproximations of resource usage in cloud programs
* compute upper bounds of computational time of concurrent programs (with synchronizations)

COST ANALYSIS

there are cost equations

linear constraints: conjunctions of $l \geq l'$ or $l = l'$ or $l \leq l'$

l is a linear expression: $k_0 + k_1 x_1 + \ldots + k_n x_n$

$$C(\mathbf{x}) = e + \textstyle\sum_{i \in 1..n} D_i(\mathbf{y}) \qquad [\varphi]$$

expression e::= k | x | e+e | nat(e-e) | e*k | e/k | max(e,e)

presburger arithmetics expressions

example: compute products of factorial

```
fact(n) = 0                  [n=0]
fact(n) = 1 + fact(n-1)      [n>0]
```

output: Maximum cost of fact(n): max([1*n,0])

A SUBSET OF SOLIDITY

```
Programs ::= (Contract)* Body

Contract ::= contract C {
                 Variables
                 Functions
                 [ function () payable { } ]      // fallback function: empty body!
                 [ constructor (T x) public { Body } ]
             }

Variables ::= (T x ;)*

Functions ::= (function f (T x) (payable)? { Body })*      // no return value!

T ::= uint | bool // no Address type!

Body ::= ( Stm )+

Stm ::=  x = E ; | E.f[.value(E)](E) ; | if (E){ Stm } else { Stm }
      |  E.transfer(E) ; | revert() ;                          // no return statement!

E ::= n | true | false x | this | msg.sender | msg.value | E.balance | E # E | E op
      | - E  | !E

# ::= + | - | > | = | ≥ | ≤ | && | ||                    op ::=  *k | /k
```

AN EXAMPLE: THE THIEF CODE

```
contract Bank {
    function pay(uint n) payable {
       if ((msg.value >= 1) && (this.balance >= n)){
          msg.sender.transfer(n) ;
          msg.sender.ack() ;
         } else { n = n ; }
     }
    function init(){
          Thief.ack() ;
    }
    function() payable { }
}
contract Thief {
    function ack() {
       msg.sender.pay.value(1)(2) ;
    }
    function() payable { }
}
balance = 101 ;
Bank.transfer(100) ;
Thief.transfer(1) ;
Bank.init() ;
```

you can drain the whole bank account!

A QUICK DEMO

FUTURE WORK

we address a very basic subset of Solidity

1. we must consider addresses and mappings
2. continuations
3. extend the compiler to CoFloCo

THE END