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)
there are cost equations

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

expression \( e ::= k \mid x \mid e+e \mid \text{nat}(e-e) \mid e*k \mid e/k \mid \text{max}(e,e) \)

**example:** compute products of factorial

\[
\begin{align*}
\text{fact}(n) &= 0 \quad \text{[n=0]} \\
\text{fact}(n) &= 1 + \text{fact}(n-1) \quad \text{[n>0]} 
\end{align*}
\]

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

1 contract Bank {
2   function pay(uint n) payable {
3       if (((msg.value >= 1) && (this.balance >= n)) {
4           msg.sender.transfer(n);
5           msg.sender.ack();
6       } else { n = n; }
7     }
8     function init() {
9         Thief.ack();
10     }
11     function() payable { }
12 }

13 contract Thief {
14   function ack() {
15       msg.sender.pay.value(1)(2); 
16   }
17   function() payable { }
18 }

19 balance = 101;
20 Bank.transfer(100);
21 Thief.transfer(1);
22 Bank.init();

you can drain the whole bank account!
A QUICK DEMO
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