Visualizing CHC Verification Conditions for Smart Contract Auditing - CHCViz

MARCO DI IANNI[1,2] FABIO FIORAVANTI[1]
GIULIA MATRICARDI[1,2]

(1) UNICH – *University of Chieti-Pescara*
(2) Dottorato Nazionale Blockchain e Distributed Ledger Technologies
UNICAM – *University of Camerino*

SMART CONTRACT

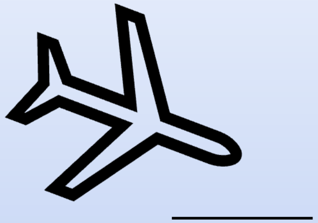
Company Airline

Flight to Turin:
expected arrival: 12:00

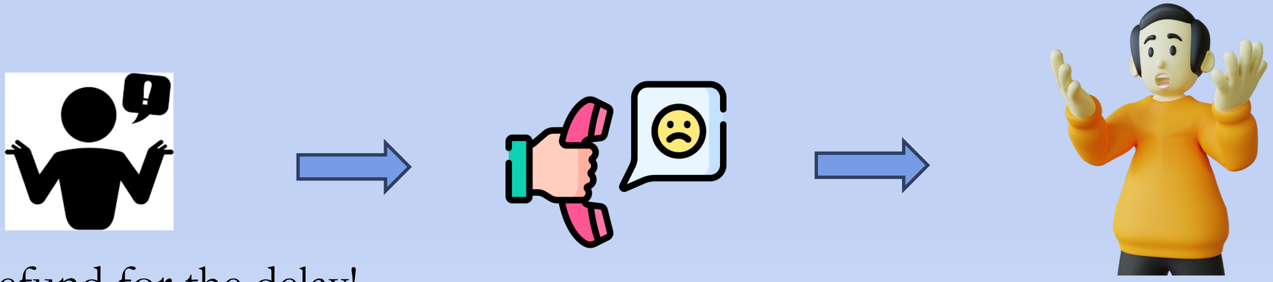
SMART CONTRACT

Flight to Turin:
expected arrival: 12:00

Company Airline



Real arrival: 14:30
Delay: 2 h 30



I want a refund for the delay!

SMART CONTRACT

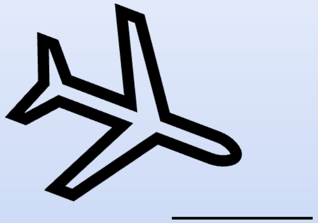
Flight insurance
Smart contract

Flight to Turin:
expected arrival: 12:00

SMART CONTRACT

Flight insurance
Smart contract

Flight to Turin:
expected arrival: 12:00



Real arrival: 14:30
Delay: 2 h 30

Automated ✓
Refund!

SMART CONTRACT

Self-executing code that enforces and executes agreements automatically when conditions are met.

Fields of application:

- Finance and Banking
- Supply Chain
- Real Estate
- Healthcare
- Legal Sector

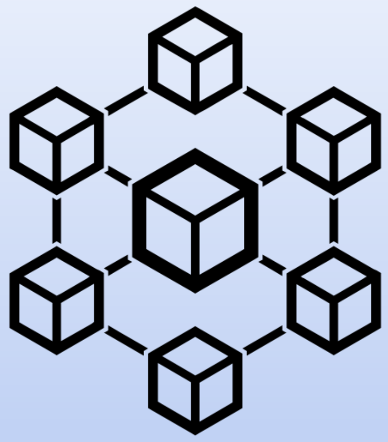
A Solidity contract

```
contract Bank {
    mapping (address => uint) balances;

    function deposit() external payable {
        uint user_balance = balances[msg.sender];
        balances[msg.sender] += msg.value;
        uint new_user_balance = balances[msg.sender];
                                                        ;
    }

    function withdraw(uint amount) public {
        require(amount > 0 && amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        (bool success,) = msg.sender.call{value: amount}("");
        require(success);
    }
}
```

SMART CONTRACT IMMUTABILITY

Smart contracts are programs acting on immutable ledgers (= libri mastri)

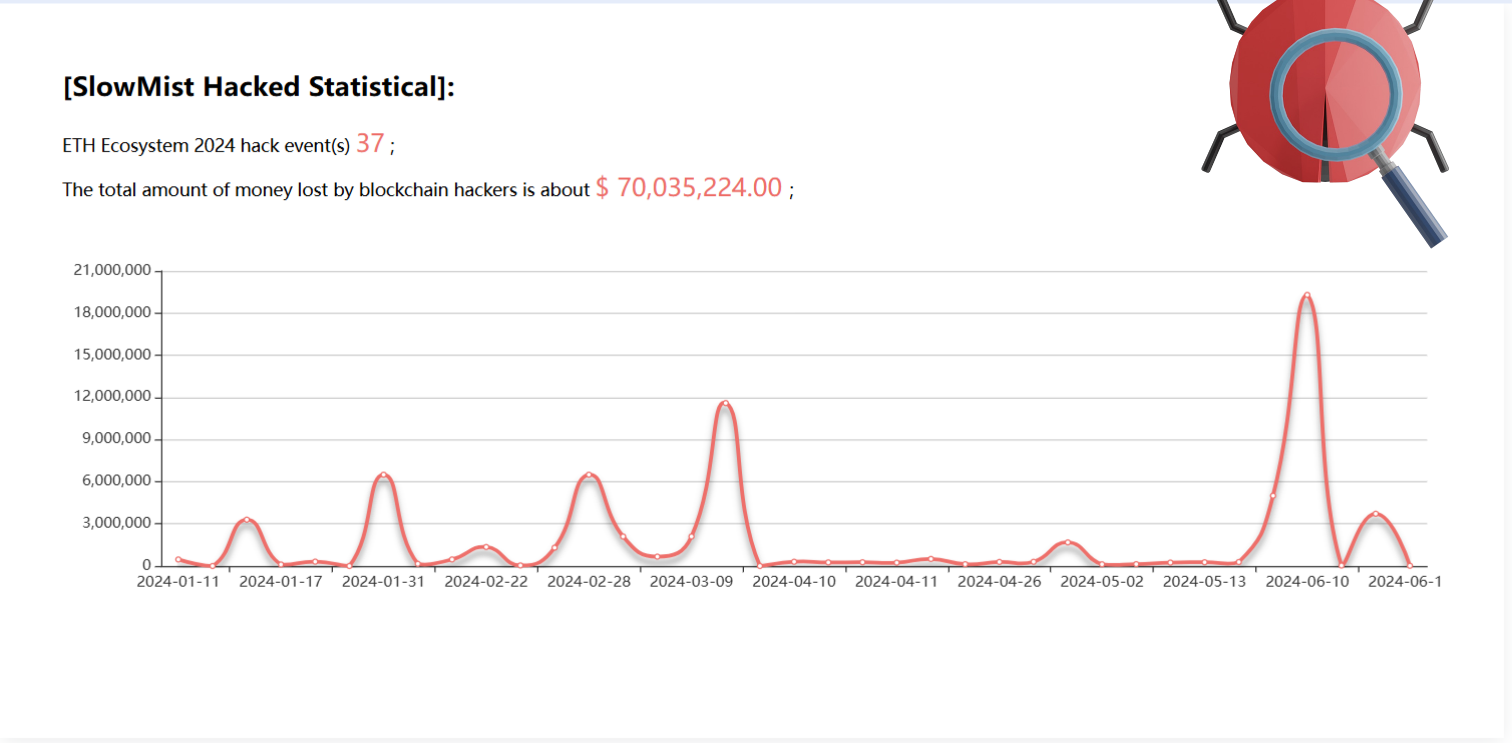
They are **immutable** and require careful consideration and proofs of the correctness properties.



SMART CONTRACT VULNERABILITIES

"Smart contracts are not free from bugs!"

- **Reentrancy (recursivity)**
- **Integer overflow**
- **Logic errors**



SMART CONTRACT AUDITING

Smart contract auditing involves evaluating code to find and mitigate vulnerabilities

Auditors techniques:

- Manual review
- Static analysis
- Symbolic execution
- Fuzzing
- Formal verification

SMART CONTRACTS FORMAL VERIFICATION

Formal Verification :

- Uses precise logic-mathematic techniques
- Proves correctness and security of the contract
- Some powerful *formal verification* methods use Constrained Horn Clauses (CHCs)

Constrained Horn Clauses (CHCs)

Constrained Horn Clauses:

- Fragment of First-Order Logic (~CLP)
- Intermediate language for program semantics
- Used in verification and program analysis
- Solvable by SMT solvers (e.g., Z3, CVC4)

```
1.  false :- M>Sum, M>=0, sum upto(M,Sum).
2.  sum upto(X,R) :- R0=0, while(X,R0,R).
3.  while(X1,R1,R) :- X1>0, R2=R1+X1, X2=X1-1, while(X2,R2,R).
4.  while(X1,R1,R) :- X1=X1=<0, R=R1.
```

CHC example for summing integers

SMART CONTRACT AUDITING WITH CHCs

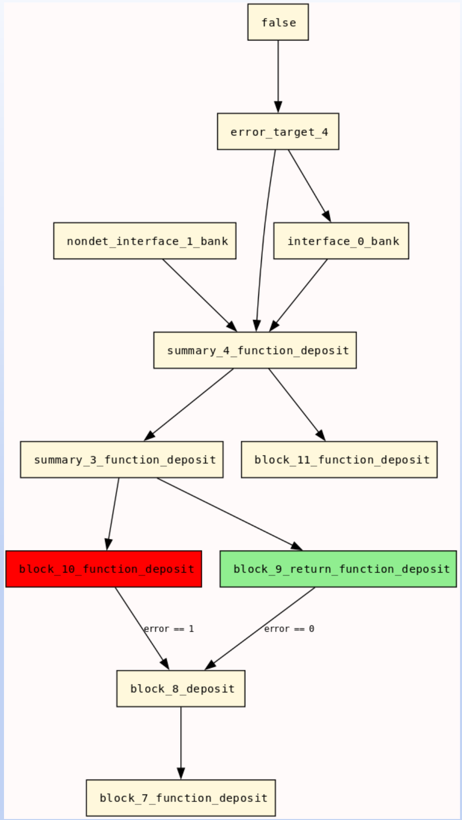
Auditing with CHCs	
Pros	**Cons**
• Use SMT solvers to prove properties • SolCMC, a module in Solidity, use CHCs	• Reading CHCs is difficult • Smart Contract in a CHCs format can contain many clauses and mutual dependencies • CHC solvers operate as a black-box

CHCViz – a tool for CHC visualization

A Solidity contract

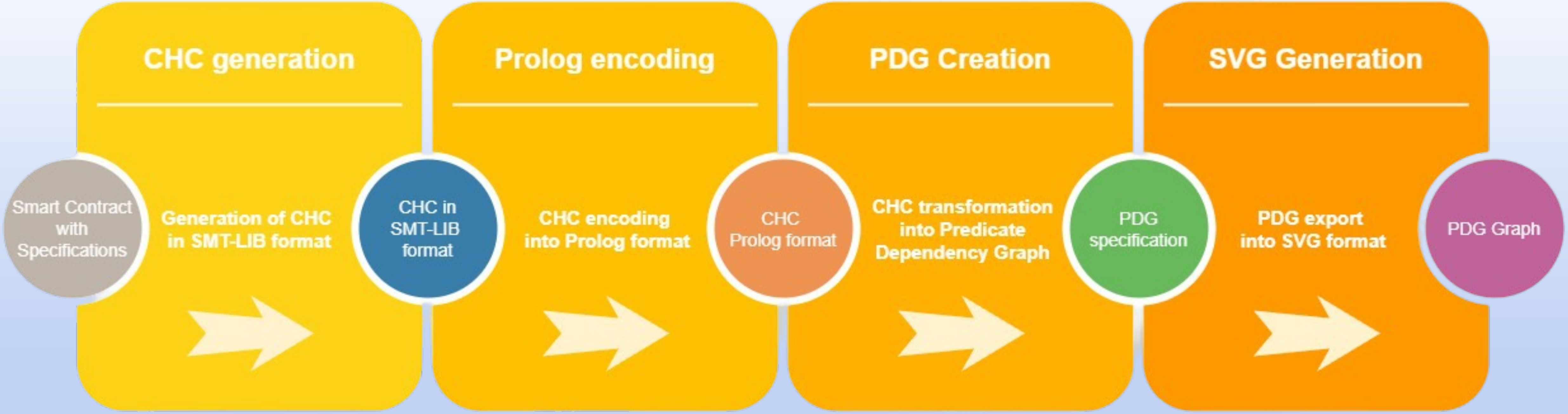
```
contract Bank {
  mapping (address => uint) balances;

  function deposit() external payable {
    uint user_balance = balances[msg.sender];
    balances[msg.sender] += msg.value;
    uint new_user_balance = balances[msg.sender];
    assert(new_user_balance == user_balance + msg.value);
  }

  function withdraw(uint amount) public {
    require(amount > 0 && amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    (bool success,) = msg.sender.call{value: amount}("");
    require(success);
  }
}
```

→ *CHCViz* →



With a **CHC visualization tool**, auditors can navigate through complex CHC predicate structures, aiding in the verification process of smart contracts.

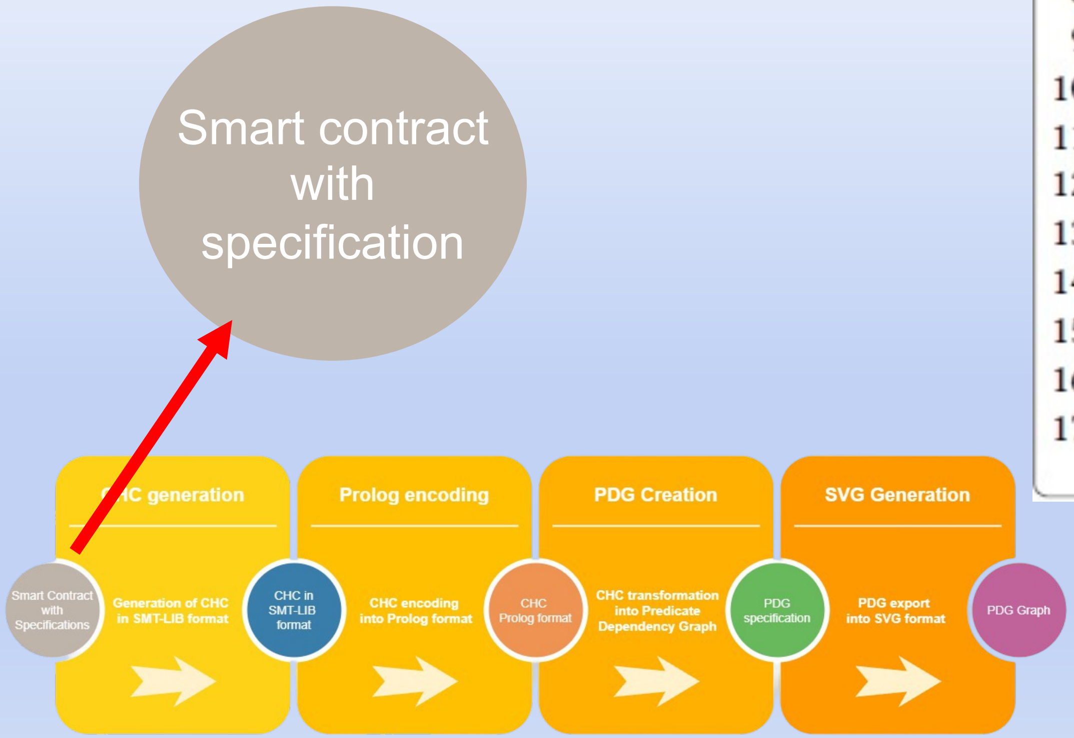
CHCViz – a tool for CHC visualization



ChcViz architecture

CHCViz

INPUT
a Solidity smart contract annotated
with pre-conditions (require)
and post-conditions (assert)



A Solidity contract

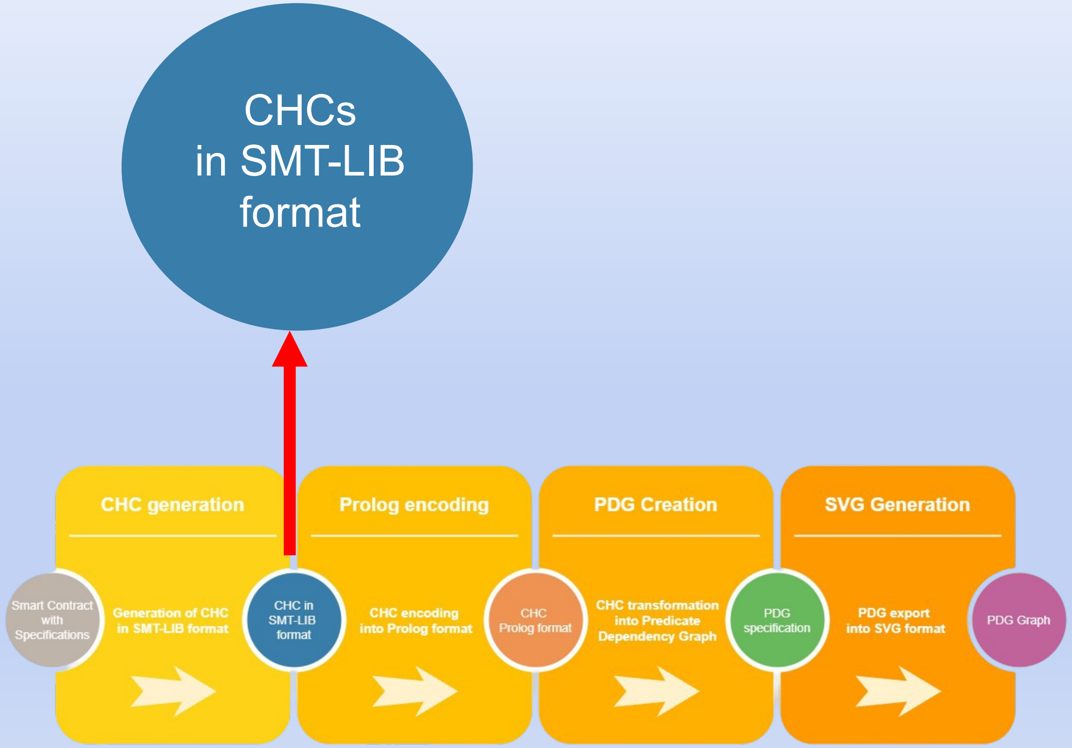
```
contract Bank {
   mapping (address => uint) balances;

   function deposit() external payable {
      uint user_balance = balances[msg.sender];
      balances[msg.sender] += msg.value;
      uint new_user_balance = balances[msg.sender];
      assert(new_user_balance == user_balance + msg.value);
   }

   function withdraw(uint amount) public {
      require(amount > 0 && amount <= balances[msg.sender]);
      balances[msg.sender] -= amount;
      (bool success,) = msg.sender.call{value: amount}("");
      require(success);
   }
}
```

Bank Smart Contract in Solidity

CHCViz

CHC GENERATION

use SolCMC to generate CHCs in SMT-LIB format



CHC clause extract in SMT-LIB format

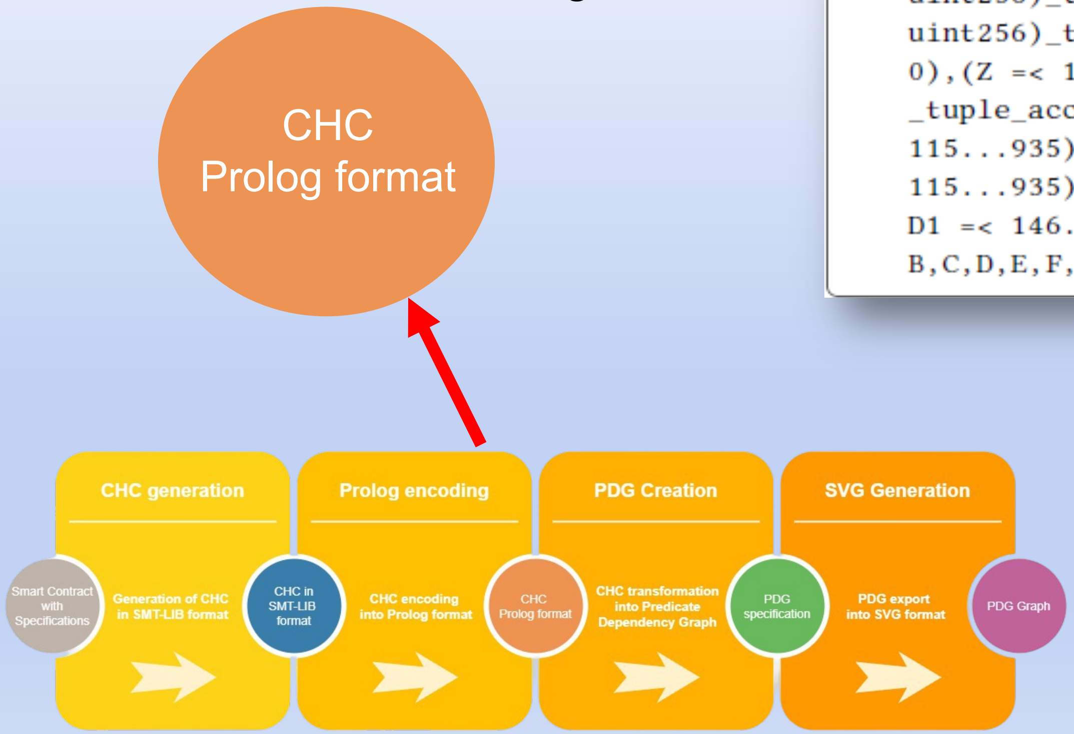
```
(declare-fun |block_10_receive_39_82_0| (Int Int |abi_type| |crypto_type| |tx_type| |state_type| |mapping(address
    => uint256)_tuple| |state_type| |mapping(address => uint256)_tuple| Int Int ) Bool)
(assert
(forall ((abi_0 |abi_type|) (amount_41_0 Int) (amount_41_1 Int) (balances_4_length_pair_0 |mapping(address =>
    uint256)_tuple|) (balances_4_length_pair_1 |mapping(address => uint256)_tuple|) (balances_4_length_pair_2 |
    mapping(address => uint256)_tuple|) (crypto_0 |crypto_type|) (error_0 Int) (error_1 Int) (error_2 Int) (
    expr_11_1 Int) (expr_12_1 Int) (expr_14_length_pair_0 |mapping(address => uint256)_tuple|) (
    expr_14_length_pair_1 |mapping(address => uint256)_tuple|) (expr_14_length_pair_2 |mapping(address => uint256)
    _tuple|) (expr_16_1 Int) (expr_17_1 Int) (expr_17_2 Int) (expr_19_1 Int) (expr_20_1 Int) (
    expr_24_length_pair_0 |mapping(address => uint256)_tuple|) (expr_26_1 Int) (expr_27_1 Int) (expr_30_0 Int) (
    expr_31_0 Int) (expr_33_1 Int) (expr_34_1 Int) (expr_35_1 Bool) (expr_9_length_pair_0 |mapping(address =>
    uint256)_tuple|) (new_user_balance_23_0 Int) (new_user_balance_23_1 Int) (new_user_balance_23_2 Int) (
    old_user_balance_8_0 Int) (old_user_balance_8_1 Int) (old_user_balance_8_2 Int) (state_0 |state_type|) (
    state_1 |state_type|) (state_2 |state_type|) (success_67_1 Bool) (this_0 Int) (tx_0 |tx_type|))
(=> (and (and (block_8__38_82_0 error_0 this_0 abi_0 crypto_0 tx_0 state_0 balances_4_length_pair_0 state_1
    balances_4_length_pair_1 old_user_balance_8_1 new_user_balance_23_1) (and (= expr_35_1 (= expr_30_0 expr_34_1)
    ) (and (=> true (and (>= expr_34_1 0) (<= expr_34_1 115...935))) (and (= expr_34_1 (+ expr_31_0 expr_33_1)) (
    and (=> true (and (>= expr_33_1 0) (<= expr_33_1 115...935))) (and (= expr_33_1 (|msg.value| tx_0)) (and (=>
    true (and (>= expr_31_0 0) (<= expr_31_0 115...935))) (and (= expr_31_0 old_user_balance_8_2) (and (=> true (
    and (>= expr_30_0 0) (<= expr_30_0 115...935))) (and (= expr_30_0 new_user_balance_23_2) (and (=
    new_user_balance_23_2 expr_27_1) (and (and (>= expr_27_1 0) (<= expr_27_1 115...935)) (and (=> true (and (>=
    expr_27_1 0) (<= expr_27_1 115...935)))
(and (= expr_27_1 (select (|mapping(address => uint256)_tuple_accessor_array| balances_4_length_pair_2) expr_26_1)
    ) (and (=> true (and (>= expr_26_1 0) (<= expr_26_1 146...975))) (and (= expr_26_1 (|msg.sender| tx_0)) (and
    (= expr_24_length_pair_0 balances_4_length_pair_2) (and (= expr_14_length_pair_2 balances_4_length_pair_2) (
    and (= balances_4_length_pair_2 (|mapping(address => uint256)_tuple| (store (|mapping(address => uint256)
    _tuple_accessor_array| expr_14_length_pair_1) expr_16_1 expr_20_1) (|mapping(address => uint256)
    _tuple_accessor_length| expr_14_length_pair_1))) (and (=> true (and (>= expr_17_2 0) (<= expr_17_2 115...935))
    ) (and (= expr_17_2 (select (|mapping(address => uint256)_tuple_accessor_array| expr_14_length_pair_1)
    expr_16_1)) (and (= expr_14_length_pair_1 balances_4_length_pair_1) (and (=> true (and (>= expr_20_1 0) (<=
    expr_20_1 115...935))) (and (= expr_20_1 (+ expr_17_1 expr_19_1)) (and (and (>= expr_17_1 0) (<= expr_17_1
    115...935)) (and (=> true (and (>= expr_17_1 0) (<= expr_17_1 115...935))) (and (= expr_17_1 (select (|mapping
    (address => uint256)_tuple_accessor_array| balances_4_length_pair_1) expr_16_1)) (and (=> true (and (>=
    expr_16_1 0) (<= expr_16_1 146...975))) (and (= expr_16_1 (|msg.sender| tx_0)) (and (= expr_14_length_pair_0
    balances_4_length_pair_1) (and (=> true (and (>= expr_19_1 0) (<= expr_19_1 115...935))) (and (= expr_19_1 (|
    msg.value| tx_0)) (and (= old_user_balance_8_2 expr_12_1) (and (and (>= expr_12_1 0) (<= expr_12_1 115...935))
     (and (=> true (and (>= expr_12_1 0) (<= expr_12_1 115...935))) (and (= expr_12_1 (select (|mapping(address =>
     uint256)_tuple_accessor_array| balances_4_length_pair_1) expr_11_1)) (and (=> true (and (>= expr_11_1 0) (<=
    expr_11_1 146...975))) (and (= expr_11_1 (|msg.sender| tx_0)) (and (= expr_9_length_pair_0
    balances_4_length_pair_1) (and (= new_user_balance_23_1 0) (and (= old_user_balance_8_1 0) true)))))))))))))
    )))))))))))))))))))))))) (and (and true (not expr_35_1)) (= error_1 1))) (block_10_receive_39_82_0 error_1
    this_0 abi_0 crypto_0 tx_0 state_0 balances_4_length_pair_0 state_1 balances_4_length_pair_2
    old_user_balance_8_2 new_user_balance_23_2))))
```

CHC clause in SMT-Lib format
of the bank smart contract

CHCViz

PROLOG ENCODING

use Eldarica, a CHC solver,
for converting CHCs
from SMT-LIB to Prolog format



CHC clause extract in Prolog format

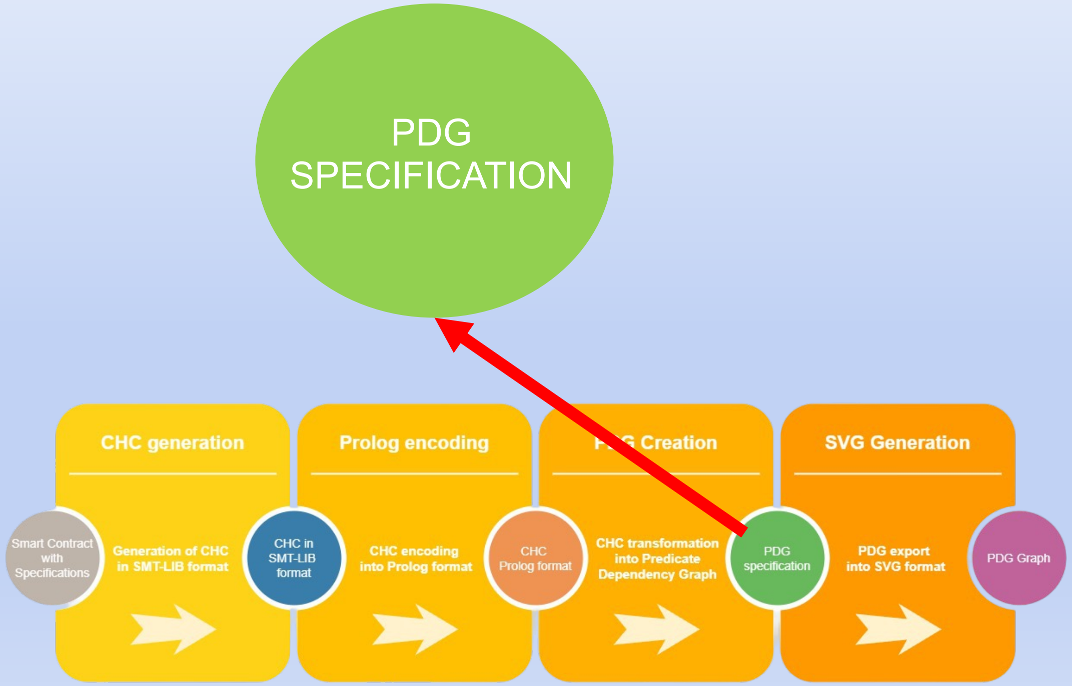
```
block_10_function_deposit(A,B,C,D,E,F,G,H,I,J,K) :- \+(L),(A = 1),\+(((L; (M = N)), (\+((M = N));
    \+(L)))),(N =< 115...935),(N >= 0),(N = (O + P)),(P =< 115...935),(P >= 0),(P = msg.value(E)
   ),(O =< 115...935),(O >= 0),(O = J),(M =< 115...935),(M >= 0),(M = K),(K = Q),(Q >= 0),(Q =<
   115...935),(Q >= 0),(Q = select(mapping(address => uint256)_tuple_accessor_array(I), R)),(R
   =< 146...975),(R >= 0),(R = msg.sender(E)),(S = I),(T = I),(I = mapping(address => uint256)
   _tuple(store(mapping(address => uint256)_tuple_accessor_array(U), V, W), mapping(address =>
   uint256)_tuple_accessor_length(U))),(X =< 115...935),(X >= 0),(X = select(mapping(address =>
   uint256)_tuple_accessor_array(U), V)),(U = Y),(W =< 115...935),(W >= 0),(W = (Z + A1)),(Z >=
   0),(Z =< 115...935),(Z =< 115...935),(Z >= 0),(Z = select(mapping(address => uint256)
   _tuple_accessor_array(Y), V)),(V =< 146...975),(V >= 0),(V = msg.sender(E)),(B1 = Y),(A1 =<
   115...935),(A1 >= 0),(A1 = msg.value(E)),(J = C1),(C1 >= 0),(C1 =< 115...935),(C1 =<
   115...935),(C1 >= 0),(C1 = select(mapping(address => uint256)_tuple_accessor_array(Y), D1)),(
   D1 =< 146...975),(D1 >= 0),(D1 = msg.sender(E)),(E1 = Y),(F1 = 0),(G1 = 0),block_8_deposit(H1,
   B,C,D,E,F,G,H,Y,G1,F1).
```

CHC clause prolog format of
the Bank smart contract

CHCViz

GRAPHIC SPECIFICATION of the PREDICATE DEPENDENCY GRAPH

using LogTalk and SWI-Prolog, logic programming languages



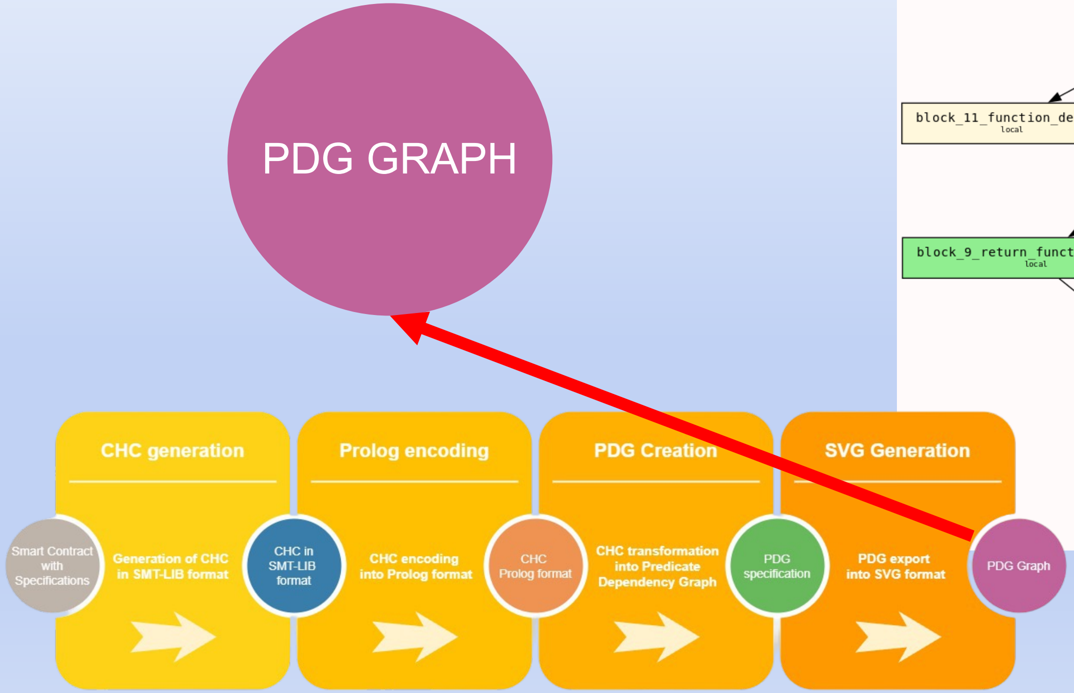

```
digraph "Bank_parsed_object" {
rankdir="TB"
ranksep="1.0"
compound="true"
splines="true"
clusterrank="local"
labeljust="l"
margin="1.0"
fontname="Monospace"
fontsize="10"
fontcolor="dimgray"
pencolor="dimgray"
stylesheet="diagrams.css"
node [shape="ellipse",style="filled",fillcolor="white",fontname="Monospace",fontsize
edge [fontname="Monospace",fontsize="9"]
label="Cross-referencing diagram for object Bank_parsed\lGenerated on 2024-05-02, 1
subgraph "cluster_Bank_parsed_object" {
bgcolor="snow"
style="rounded"
margin="10"
label=<<TABLE border="0" cellborder="0"><TR><TD tooltip="GITHUB/SmartContractToGraph
tooltip="GITHUB/SmartContractToGraph/Bank_parsed.pl"
"nondet_interface_1_bank_80_0/8" [shape="box",tooltip="GITHUB/SmartContractToGraph/
"block_7_function_deposit__39_80_0/11" [shape="box",tooltip="GITHUB/SmartContractTo
"block_8_deposit_38_80_0/11" [shape="box",tooltip="GITHUB/SmartContractToGraph/Bank_
"block_10_function_deposit__39_80_0/11" [shape="box",tooltip="GITHUB/SmartContractT
"summary_3_function_deposit__39_80_0/9" [shape="box",tooltip="GITHUB/SmartContractT
"block_9_return_function_deposit__39_80_0/11" [shape="box",tooltip="GITHUB/SmartCon
"block_11_function_deposit__39_80_0/11" [shape="box",tooltip="GITHUB/SmartContractT
"summary_4_function_deposit__39_80_0/9" [shape="box",tooltip="GITHUB/SmartContractT
"interface_0_bank_80_0/5" [shape="box",tooltip="GITHUB/SmartContractToGraph/Bank_pa
```

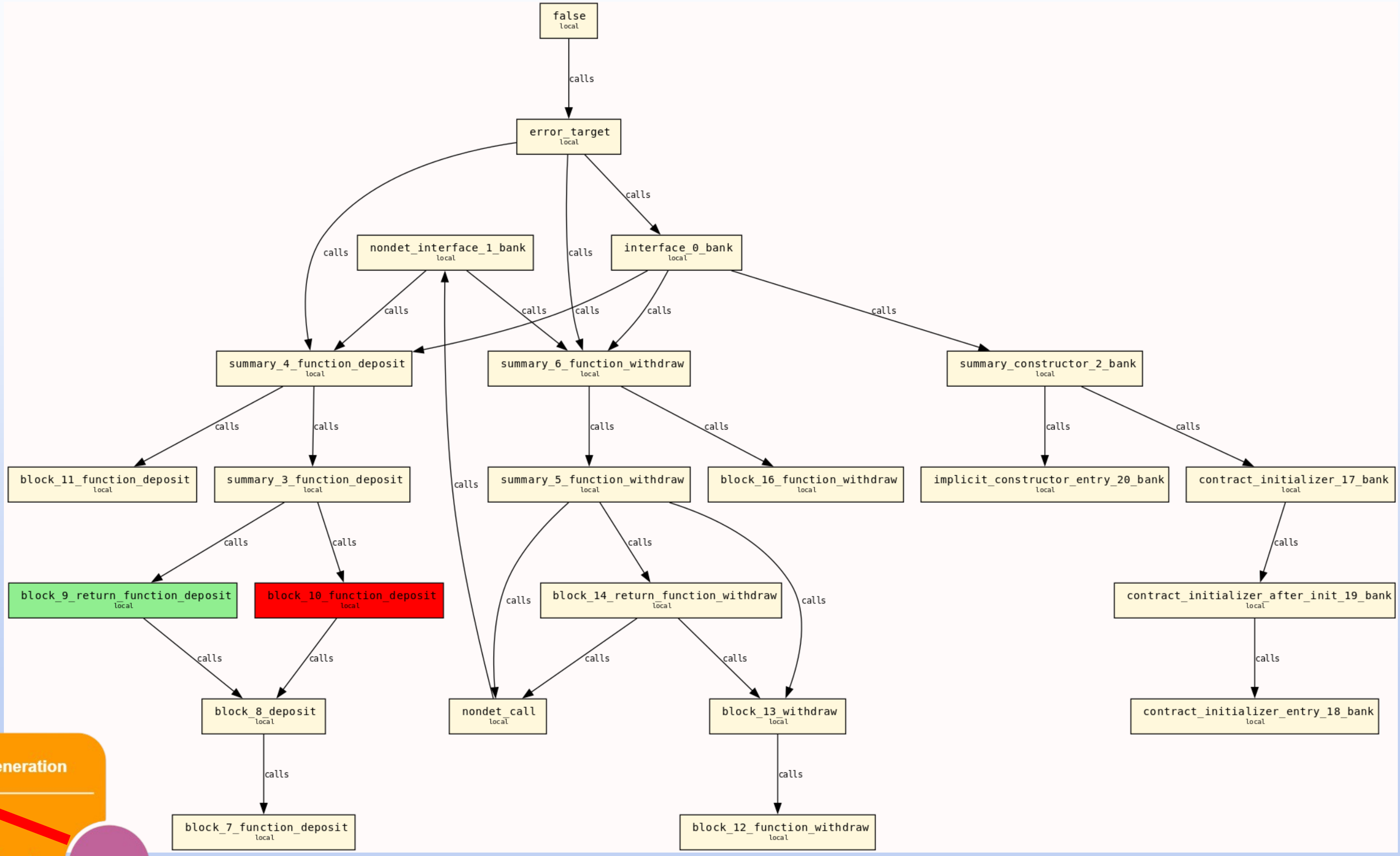
Predicate dependency Graph in .dot format

CHCViz

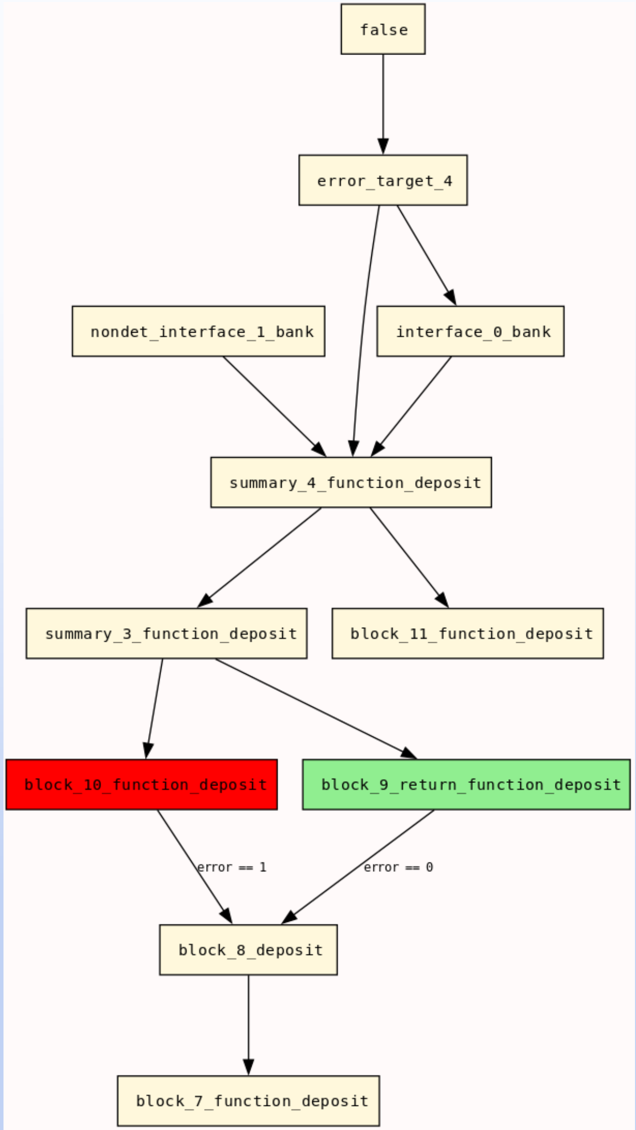
PDG GENERATION in SVG format

using GraphViz





Predicate dependency Graph in SVG



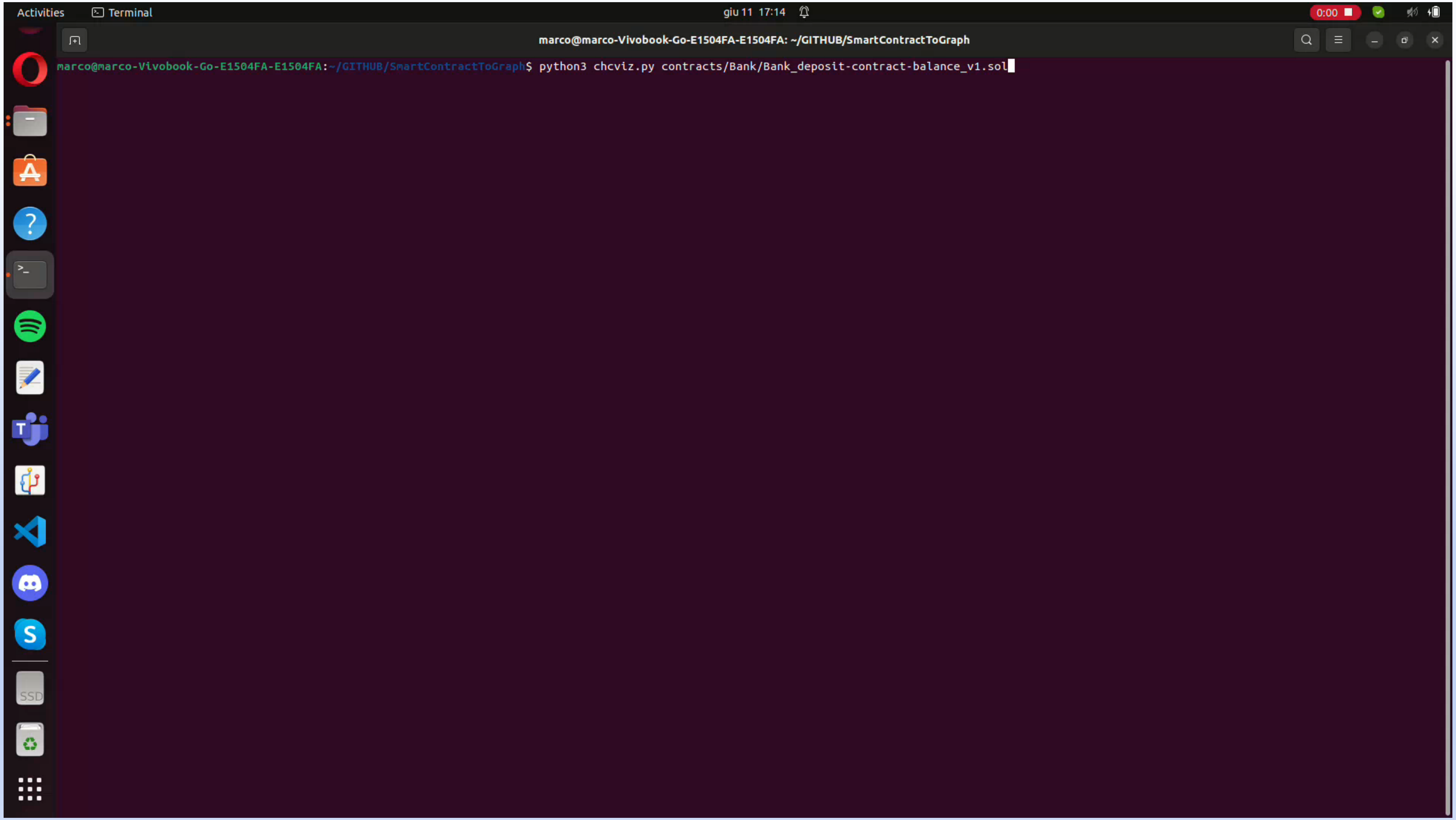
Bank deposit function PDG

PDG Benefits

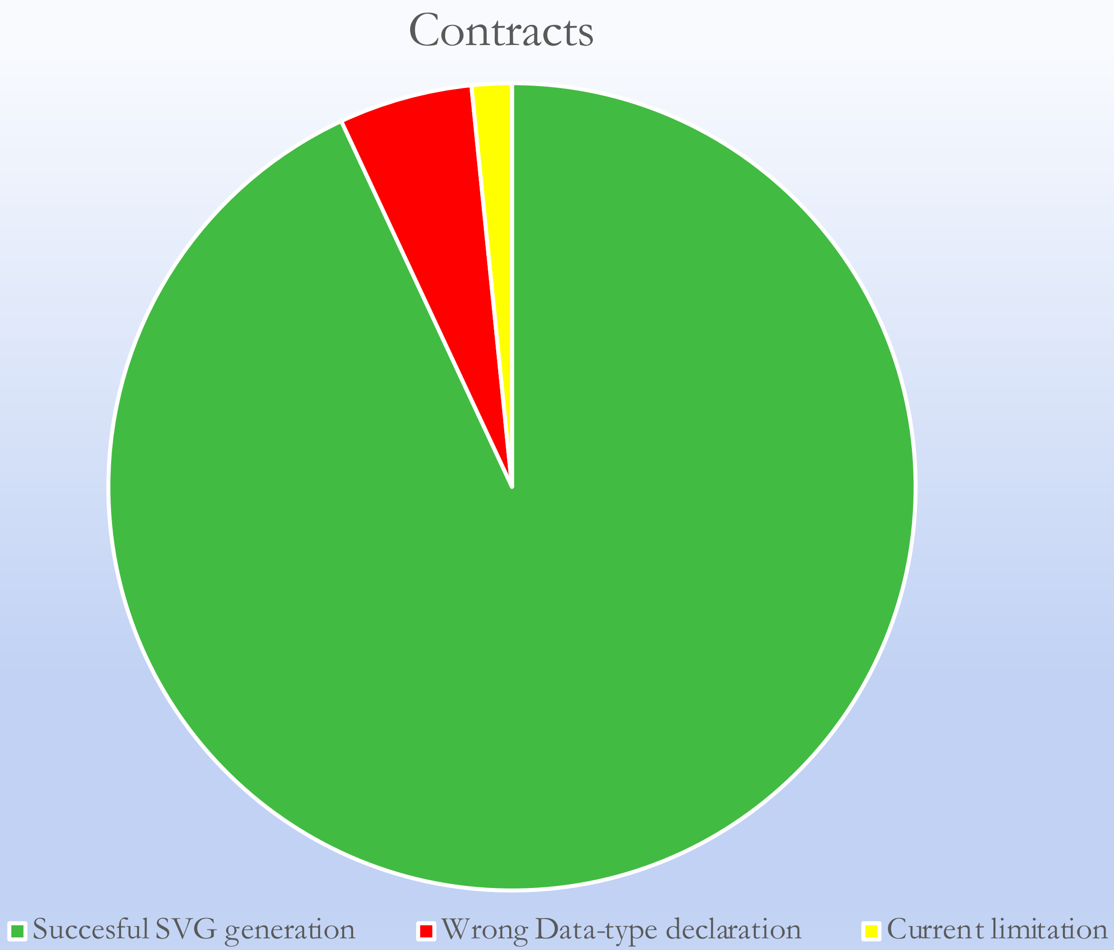
Benefits of the visual PDG:

- Dependencies among CHCs predicates are in clear
- CHCs that affects verification query are easily identifiable
- CHCs transformation can be applied to blocks directly from the PDG (future work)

CHCVIZ in practice!



Experimental results



187 contracts from Bartoletti et al. *Towards benchmarking of Solidity verification tools*, FMBC’24

- 174 successful SVG generation
- 10 data type declaration errors by SolCMC (after manual correction SVG was successfully generated)
- 3 unable to generate the '.dot' file due to a temporary limitation of our tool

CONCLUSIONS
& FUTURE WORK

CHCviz can help auditors detect specification violations and improve the reliability of Smart Contracts.

Future Work:

- Extend to other smart contract languages
- Rewrite the Prolog Encoding and the PDG Creation modules.
- Allow interaction with the PDG (e.g. expanding, collapsing, browsing) and the CHCs (program transformations, CHC solving)

See more and test CHCViz on:
https://fmlab.unich.it/chcviz

Thanks!

Happy to answer all your questions