SOLGREP

A GRAMMAR AWARE SOLIDITY QUERY TOOL

AUTHOR: **FERRAN CELADES I PONS**

YEAR: **2022**

TUTOR: **ALBERTO BALLESTEROS RODRÍGUEZ**

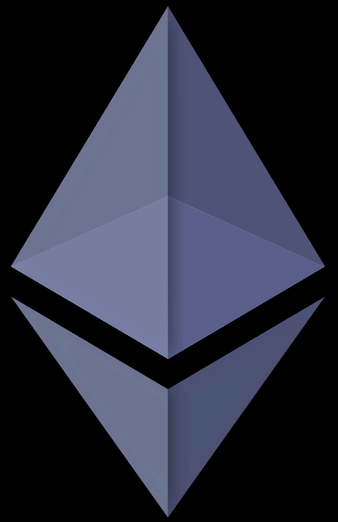
Master's degree in Information and Communication Technologies Security

Protocols and security applications

Universitat Oberta de Catalunya

01

INTRODUCTION



State of Art

Money into the space

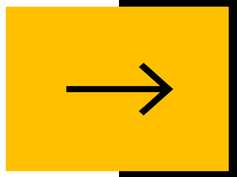
What is the problem?

Smart Contracts are public

Code Vulnerabilities

Same mistakes

Easy to contribute tools to find vulnerabilities



What is the solution?

Code Vulnerabilities

Easy to contribute Public rules

The goals

01

Parse

Solidity Code

02

Query

Using simple rules

03

Report

The finding to the

auditor and developer

04

Help

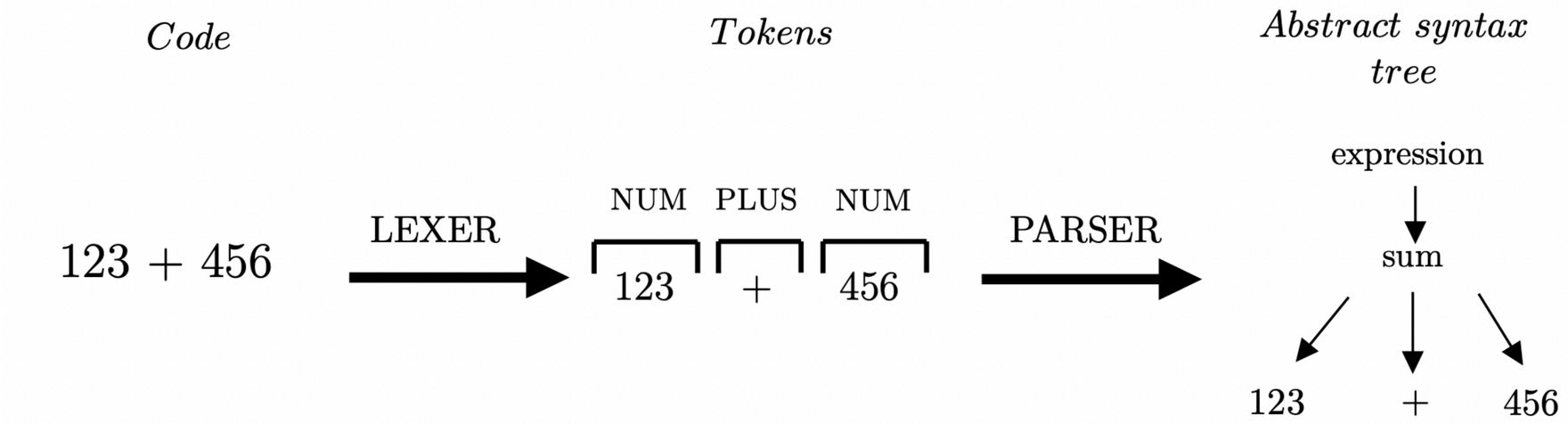
The community

02
PARSING

Solidity

```
pragma solidity ^0.7.0;

contract HelloWorld {

   string public message;

   constructor(string memory initMessage) {
      message = initMessage;
   }

   function update(string memory newMessage) public {
      message = newMessage;
   }
}
```

GRAMMAR


Code
Tokens
Abstract syntax tree
123 + 456
LEXER
NUM
PLUS
NUM
123
+
456
PARSER
expression
sum
123
+
456

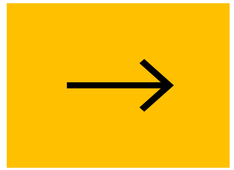
DESCRIBING GRAMMARS

```
grammar Exp;

eval
    :    additionExp
    ;

additionExp
    :    multiplyExp
         ( '+' multiplyExp
         | '-' multiplyExp
         )*
    ;

multiplyExp
    :    atomExp
         ( '*' atomExp
         | '/' atomExp
         )*
    ;

atomExp
    :    Number
    |    '(' additionExp ')'
    ;

Number
    :    ('0'..'9')+ ('.' ('0'..'9')+)?
    ;
```

```
module.exports = grammar({
  name: 'math',

  precedences: _ => [
    [
      "multiplication",
      "addition",
    ],
  ],

  rules: {
    expression: $ => $._expression,
    _expression: $ => choice(
      $.number,
      $.sum,
      $.product,
    ),

    sum: $ => prec.left(
      "addition",
      seq(
        field("left", $._expression),
        "+",
        field("right", $._expression),
      ),
    ),

    product: $ => prec.left(
      "multiplication",
      seq(
        field("left", $._expression),
        "*",
        field("right", $._expression),
      ),
    ),
    number: _ => /\d+(\.\d+)?/,
  }
});
```

TREE-SITTER GRAMMAR

JoranHonig / **tree-sitter-solidity** Public



fr0zn / **tree-sitter-solidity** Public

Missing

Solidity features

- Experimental pragma support
- Tuple variable declaration
- Usage of var keyword to declare variables and tuples
- Slicing optional members, such as [4:]

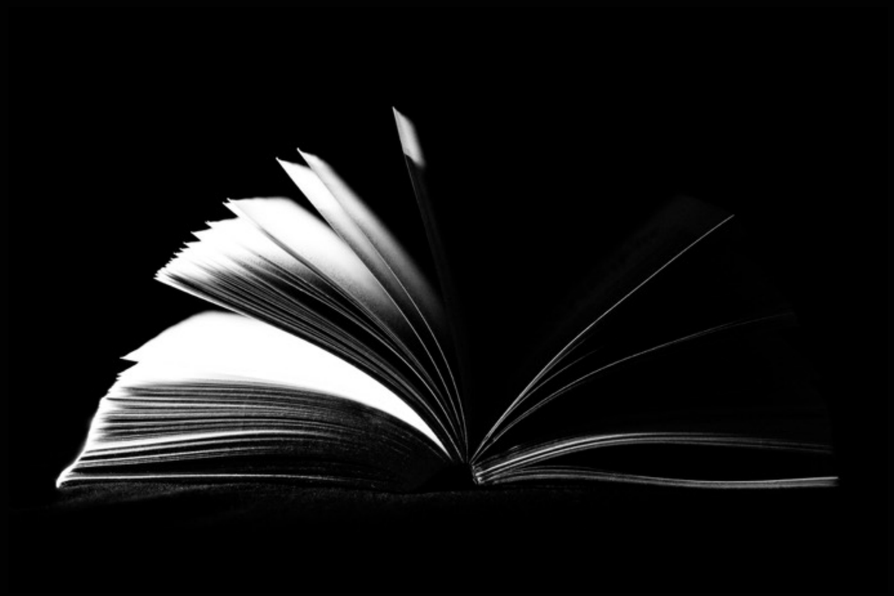
COVERAGE

Top 30 Solidity projects on Github

```
https://github.com/ethereum/EIPs
https://github.com/Aircoin-official/AirCash
https://github.com/Dapp-Learning-DAO/Dapp-Learning
https://github.com/fravoll/solidity-patterns
https://github.com/willitscale/learning-solidity
https://github.com/sushiswap/sushiswap
https://github.com/Rari-Capital/solmate
https://github.com/crytic/not-so-smart-contracts
https://github.com/compound-finance/compound-protocol
https://github.com/crytic/echidna
https://github.com/nibbstack/erc721
https://github.com/ExtropyIO/defi-bot
https://github.com/xtblock/xtt
https://github.com/solidlyexchange/solidly
https://github.com/Arachnid/solidity-stringutils
https://github.com/studydefi/money-legos
https://github.com/PatrickAlphaC/nft-mix
https://github.com/crytic/building-secure-contracts
https://github.com/provable-things/ethereum-api
https://github.com/Uniswap/v2-periphery
https://github.com/OlympusDAO/olympus-contracts
https://github.com/safemoonprotocol/Safemoon.sol
https://github.com/yam-finance/yam-protocol
https://github.com/zeriontech/defi-sdk
https://github.com/unlock-protocol/unlock
https://github.com/andrecronje/rarity
https://github.com/aragon/aragonOS
https://github.com/HashLips/hashlips_nft_contract
https://github.com/makerdao/multicall
```

03

QUERYING

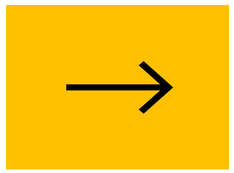


- INTERPRETING THE AST
- COMPARING A TREE
- ELLIPSIS NODE
- EXTENDING THE GRAMMAR
- METAVARS

INTERPRETING THE AST

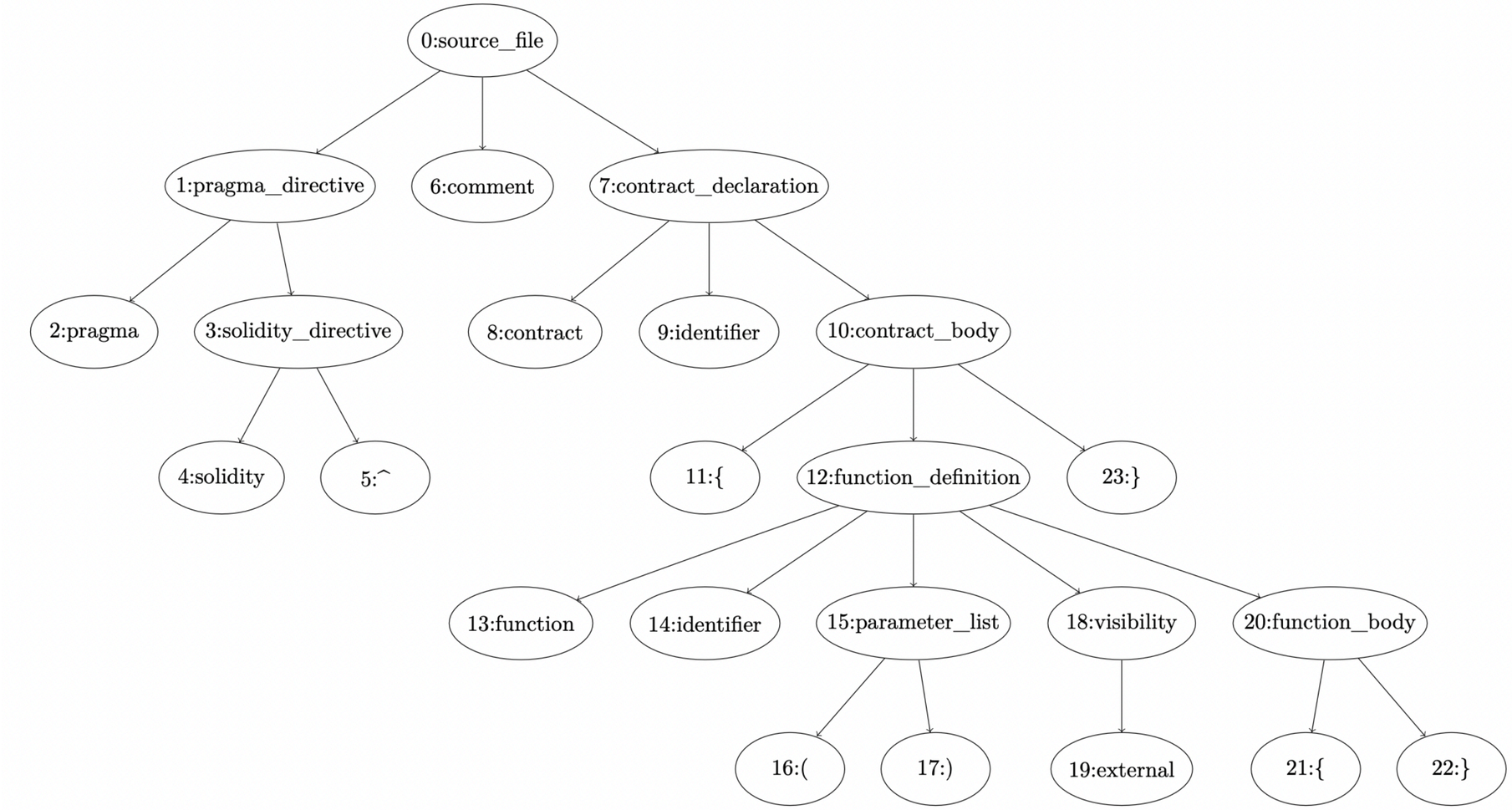
```
pragma solidity ^0.8.4;

// Comment
contract TestContract {

  function test() external {
  }
}
```

tree-sitter



```
(source_file [0, 0] - [8, 1]
  (pragma_directive [0, 0] - [0, 23]
    (solidity_directive [0, 7] - [0, 22]
      (pragma_versions [0, 16] - [0, 22])))
  (comment [3, 0] - [3, 11])
  (contract_declaration [4, 0] - [8, 1]
    name: (identifier [4, 9] - [4, 21])
    body: (contract_body [4, 22] - [8, 1]
      (function_definition [6, 2] - [7, 3]
        function_name: (identifier [6, 11] - [6, 15])
        (parameter_list [6, 15] - [6, 17])
        (visibility [6, 18] - [6, 26])
        body: (function_body [6, 27] - [7, 3])))))
```

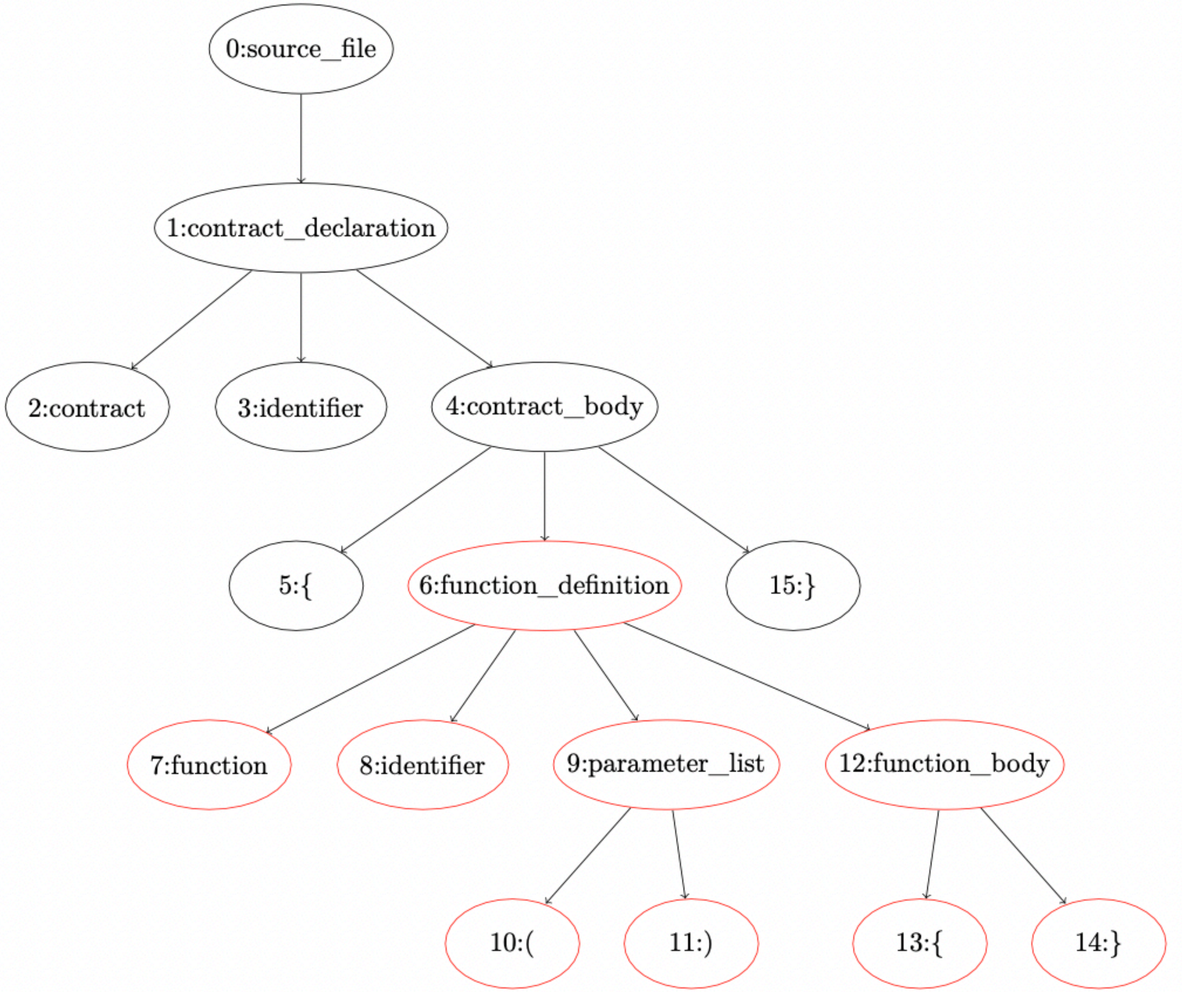
INTERPRETING THE AST



COMPARING A TREE

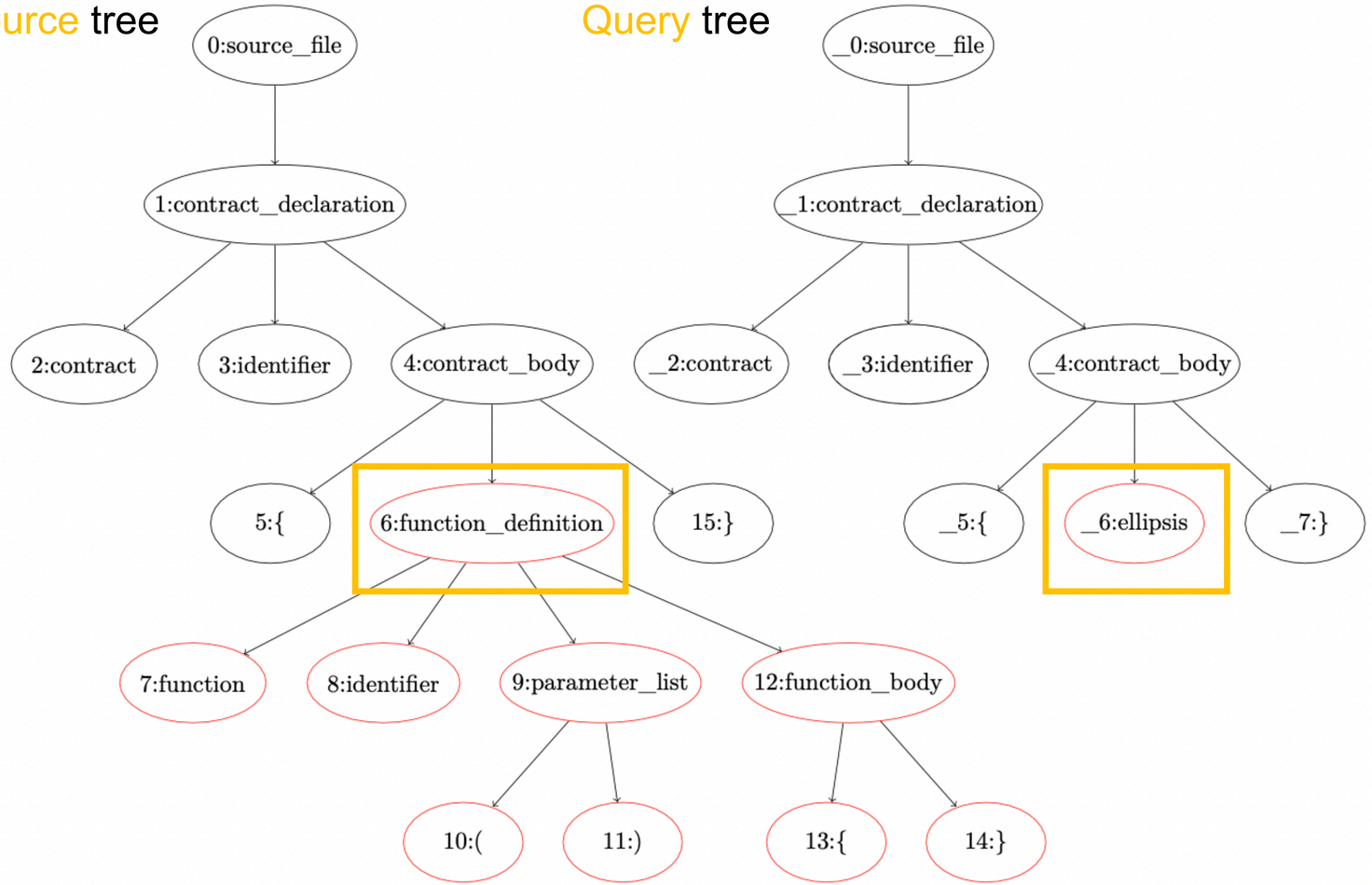
```
contract TestContract {
  function test() {

  }
}
```

Find any contract



ELLIPSIS NODE

Source tree

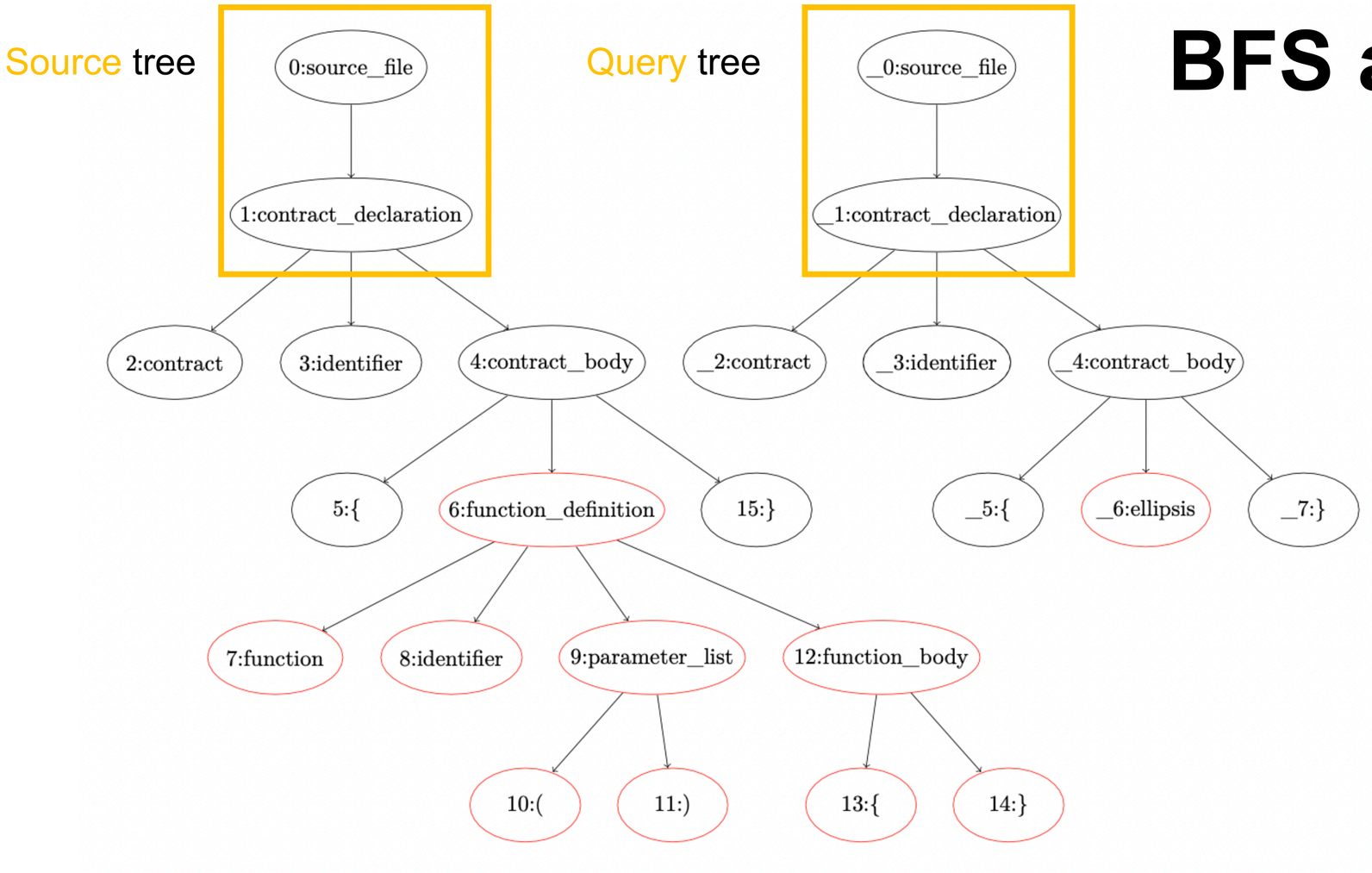


Query tree


```
contract TestContract {
  function test() {

  }
}
```

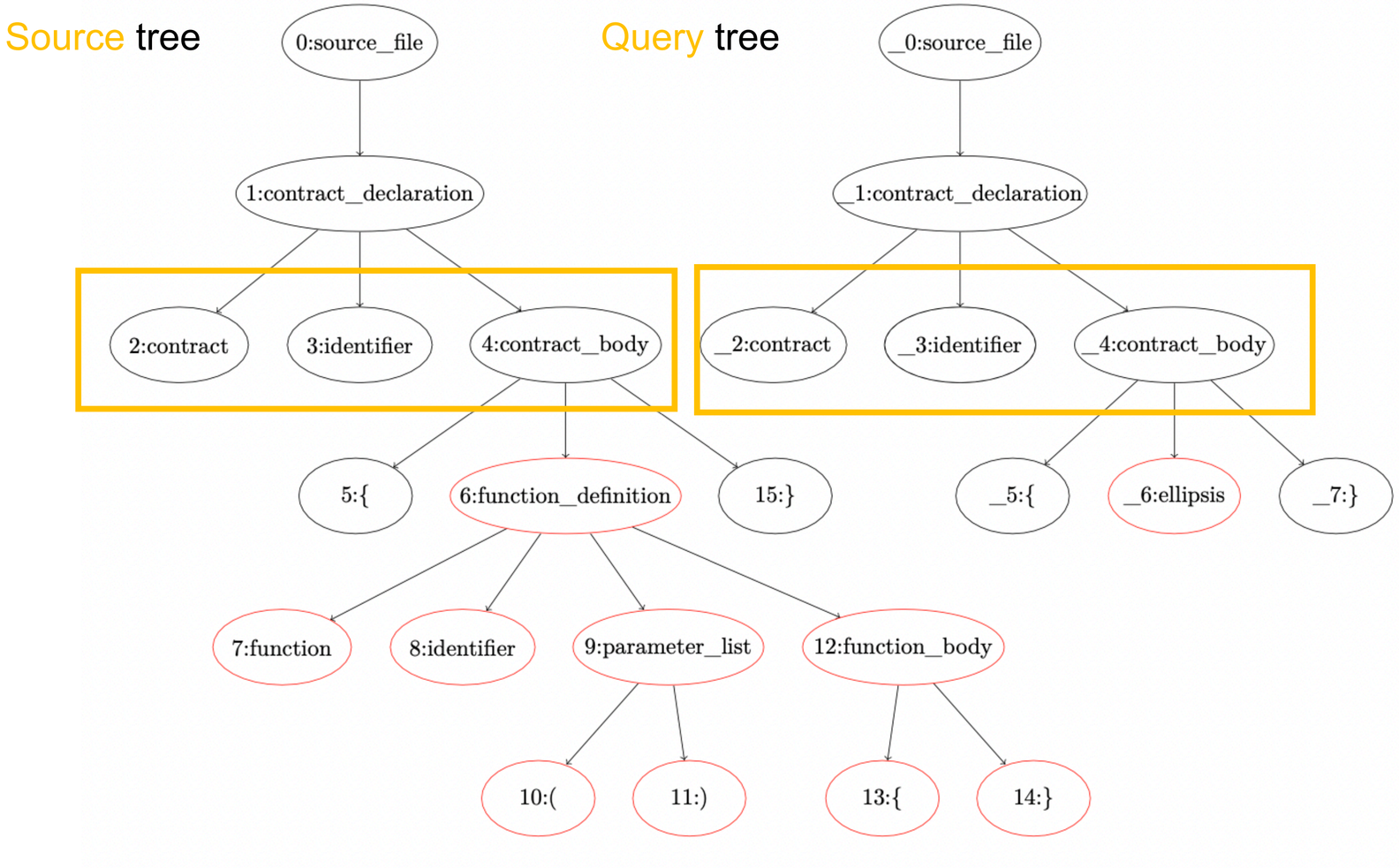
```
contract TestContract {
  ...
}
```

COMPARING A TREE

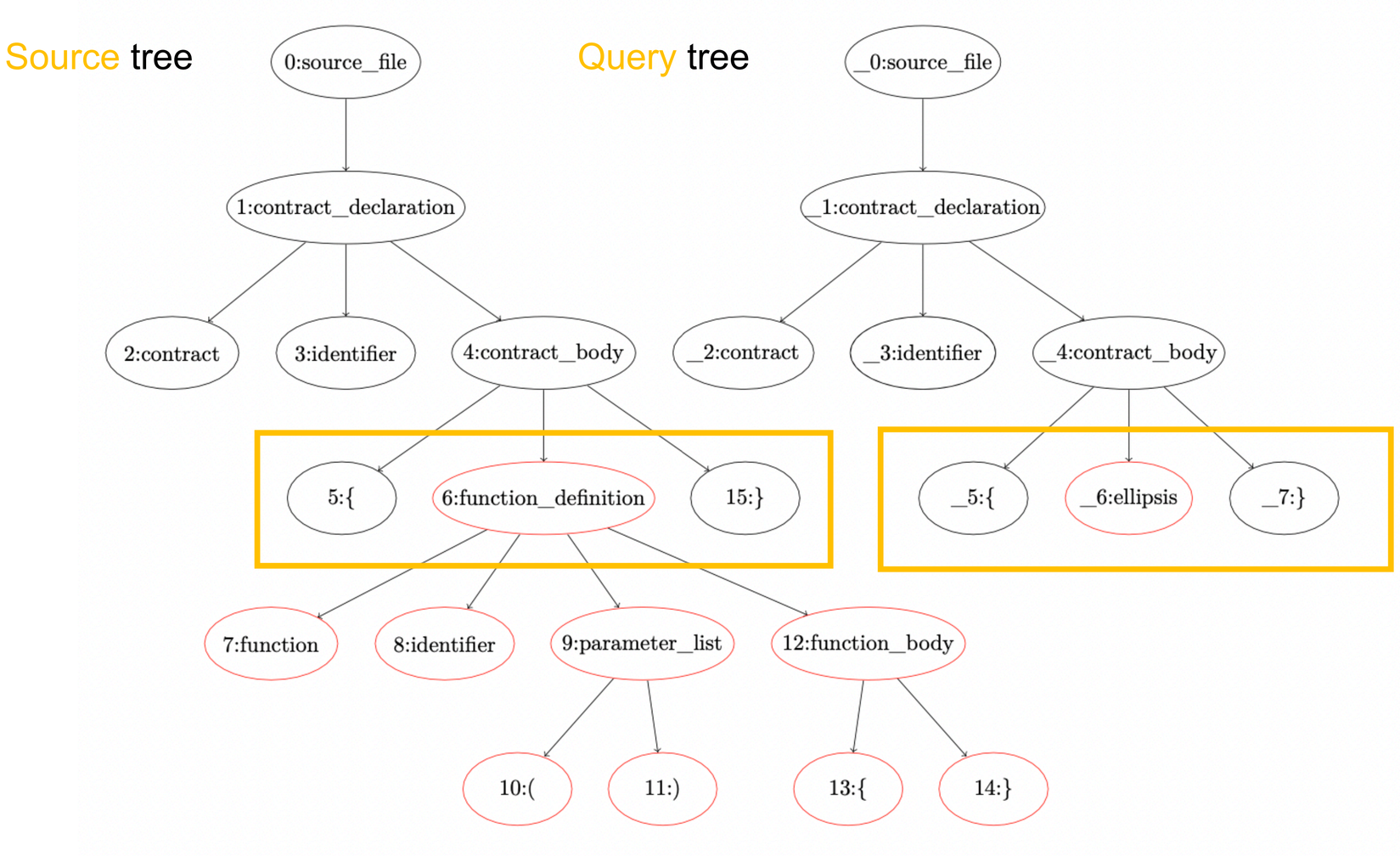


BFS algorithm

COMPARING A TREE

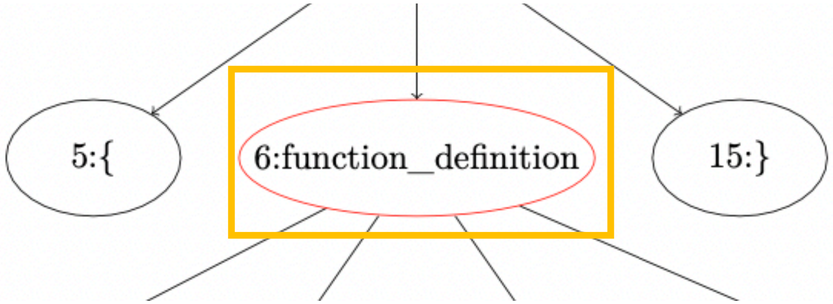


COMPARING A TREE

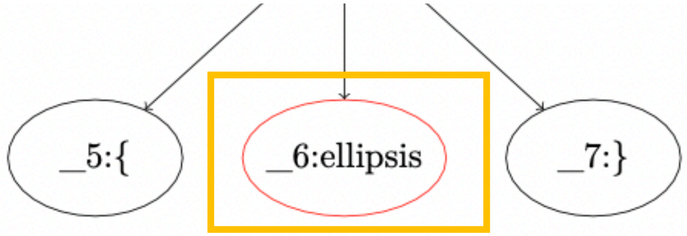


COMPARING A TREE

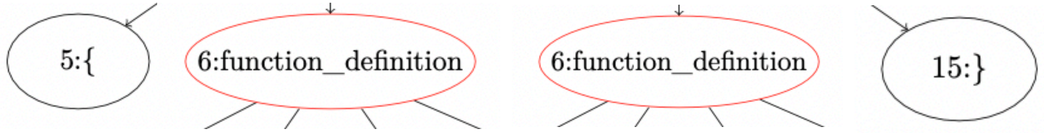
Source tree



Query tree



Ellipsis matches none or any node (similar to * on regex)




```
a = [1, 2, 3, 4, 5]
b = ['.', 2, '.', 5]
assert compare(a, b) == True

a = [1, 2, 3, 4, 5]
b = ['.', 5]
assert compare(a, b) == True

a = [1, 2, 3, 4, 5]
b = [1, '.', 5]
assert compare(a, b) == True

a = [1, 2, 3, 4, 5]
b = [1, 2, '.']
assert compare(a, b) == True

a = [1, 2, 3, 4, 5]
b = ['.']
assert compare(a, b) == True

a = [1, 2, 3, 4, 5]
b = [1, '.', '.', '.', 4, '.', '.', 5]
assert compare(a, b) == True
```

EXTENDING THE GRAMMAR

```
_contract_body: ($, previous) ⇒ {
  return choice(
    ...previous.members,
    $.ellipsis
  );
},

_expression: ($, previous) ⇒ {
  return choice(
    ...previous.members,
    $.ellipsis,
  );
},

ellipsis: $ ⇒ '...',
```

```
contract TestContract {
  ...
}
```

```
(source_file [0, 0] - [2, 1]
  (contract_declaration [0, 0] - [2, 1]
    name: (identifier [0, 9] - [0, 21])
    body: (contract_body [0, 22] - [2, 1]
      (ellipsis [1, 2] - [1, 5]))))
```

METAVARS - WHY?

```
contract ContractName {
  function func1() external {}
  function func2() external {
    func1();
  }

  function func3() external {
  }
  function func4() external {
    func3();
  }
}
```

```
contract ContractName {
   ... // match any previous code
  function ...() external {}
   ...
  function ...() external {
     ...();
  }
   ... // match any following code
}
```

METAVARS - WHY?

```
contract ContractName {
  ... // match any previous code
  function ...() external {}
  ...
  function ...() external {
    ...();
  }
  ... // match any following code
}
```

```
contract ContractName {
  function func1() external {}
  function func2() external {
    func1();
  }

  function func3() external {
  }
  function func4() external {
    func3();
  }
}
```

```
contract ContractName {
  function func1() external {}
  function func2() external {
    func1();
  }

  function func3() external {
  }
  function func4() external {
    func3();
  }
}
```

METAVARS - WHY?

```
contract ContractName {
  function func1() external {}
  function func2() external {
    func1();
  }

  function func3() external {
  }
  function func4() external {
    func3();
  }
}
```

```
contract ContractName {
   ... // match any previous code
  function ...() external {}
   ...
  function ...() external {
    ...();
  }
   ... // match any following code
}
```

METAVARS

Metavars are represented with the $ followed by an identifier

```
contract ContractName {
  function func1() external {}
  function func2() external {
    func1();
  }

  function func3() external {
  }
  function func4() external {
    func3();
  }
}
```

```
contract ContractName {
  ... // match any previous code
  function $FNC() external {
    ...;
  }
  ...
  function $CALLER() external {
    $FNC();
  }
  ... // match any following code
}
```

```
contract ContractName {
  function func1() external {}
  function func2() external {
    func1();
  }

  function func3() external {
  }
  function func4() external {
    func3();
  }
}
```

METAVARS - INTERNAL

- $TYPE: uint256, bool, …
- $VISIBILITY: public, external, internal, …
- $STATE: view, pure
- $STORAGE: memory, storage, call data
- $VERSION: any pragma version

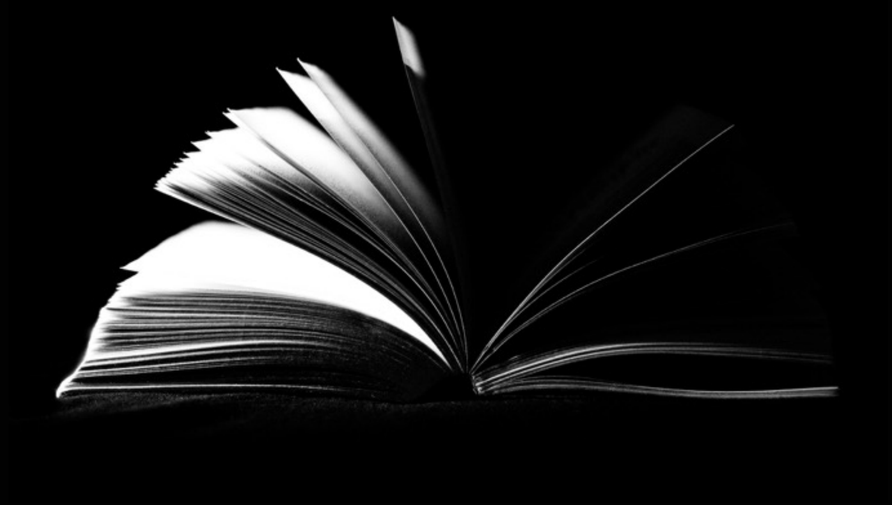
All internal metavars do support enumeration, allowing multiple metavars to be used: $TYPE1, $TYPE2 …

METAVARS - INTERNAL EXAMPLE

```
contract $CONTRACT {

  function $FNC1($TYPE0 $VAR1) $VISIBILITY returns($TYPE1){
    ...
    $TYPE1 $VAR2 = $FNC2($VAR1);
    ...
    return $VAR2;
  }
}
```

04

SOLGREP RULES



- RULES SYNTAX
- MESSAGE PLACEHOLDERS
- SOLGREP PATTERNS
- METAVARS REGEX
- REPORT OUTPUT

RULES SYNTAX


```yaml
id: issue-id
message: |
  This is the message {{CONTRACTS | comma}}
risk: 1
impact: 1
patterns:
  - pattern: contract $CONTRACT { ... }
    and:
      - pattern: function $FUN( ... ) ... { ... }
        and:
        - pattern: ... -= ...
        - pattern: ... *= ...
        - pattern: ... += ...
metavars-regex:
  $CONTRACT: .*
  $FUN: admin.*
```

id: Used to identify the rule

message: The message/description of the issue. It will be used during the reporting

risk/impact: For auditors to classify the issue severity

patterns: Solgrep expressions for the query

metavars-regex: Metavars regex comparison format

MESSAGE PLACEHOLDERS

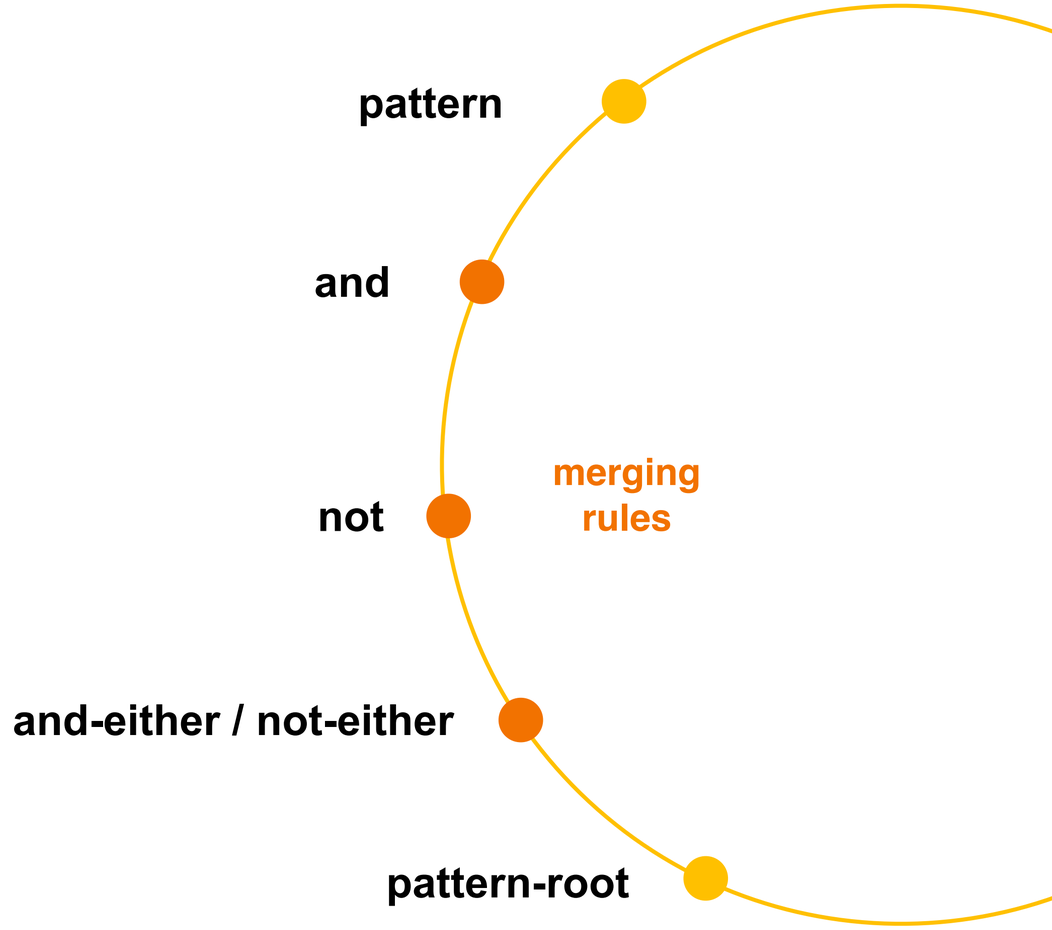
```yaml
id: issue-id
message: |
  This is the message {{CONTRACTS | comma}}
risk: 1
impact: 1
patterns:
  - pattern: contract $CONTRACT { ... }
    and:
      - pattern: function $FUN( ... ) ... { ... }
        and:
        - pattern: ... -= ...
        - pattern: ... *= ...
        - pattern: ... += ...
metavars-regex:
  $CONTRACT: .*
  $FUN: admin.*
```

METAVAR + S: It will contain metavar values for all valid query results as a list

CONTENTS: It will contain the content of the matching rule, from the start of the match to the end

SOLGREP PATTERNS

```yaml
id: issue-id
message: |
  This is the message {{CONTRACTS | comma}}
risk: 1
impact: 1
patterns:
  - pattern: contract $CONTRACT { ... }
    and:
      - pattern: function $FUN( ... ) ... { ... }
        and:
        - pattern: ... -= ...
        - pattern: ... *= ...
        - pattern: ... += ...
metavars-regex:
  $CONTRACT: .*
  $FUN: admin.*
```



METAVARS REGEX

```
id: issue-id
message: |
  This is the message {{CONTRACTS | comma}}
risk: 1
impact: 1
patterns:
  - pattern: contract $CONTRACT { ... }
    and:
      - pattern: function $FUN( ... ) ... { ... }
        and:
        - pattern: ... -= ...
        - pattern: ... *= ...
        - pattern: ... += ...
metavars-regex:
  $CONTRACT: .*
  $FUN: admin.*
```

/regex/

The metavars-regex dictates the value that the metavar will have on all identifiers comparisons

REPORT OUTPUT

id: Used to identify the rule

message: The message/description of the issue. It will be used during the reporting

risk/impact: For auditors to classify the issue severity

metavars: List of valid metavars of the ones used in the query

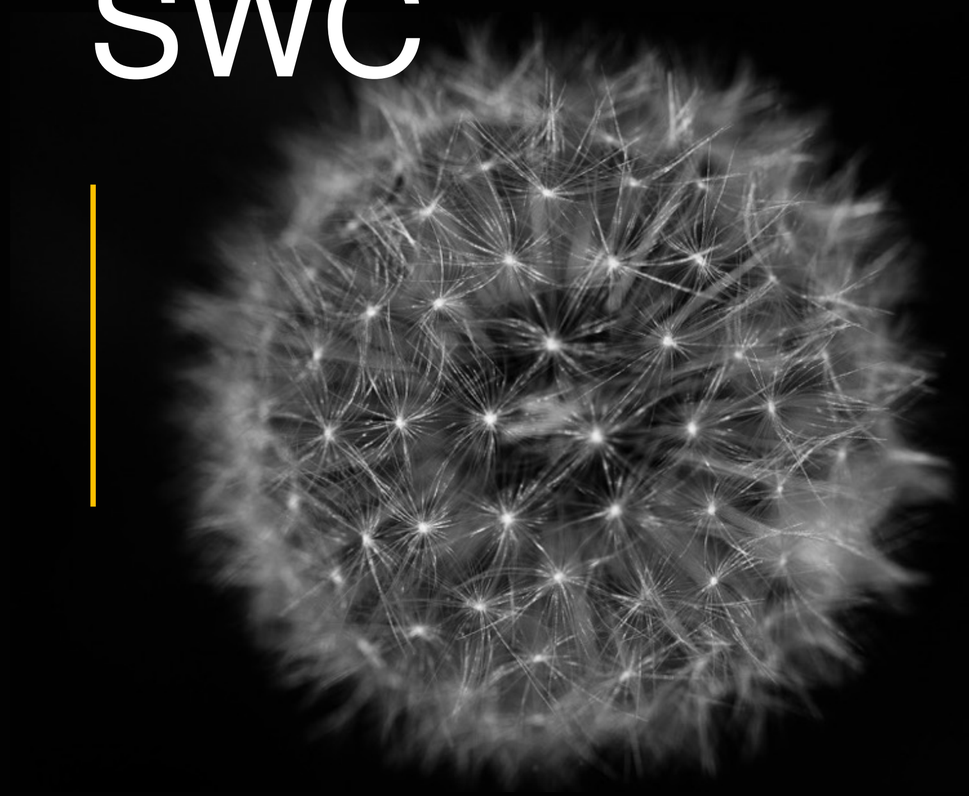
bytes range: Does contain the start and end byte of the matching query on the original source code

lines range: Does contain the start line and character at that line and the end line and character of the matching query on the original source code


```
{
  "id": "solidity-test",
  "message": "Found a function: name",
  "risk": 1,
  "impact": 1,
  "results": 1,
  "metavars": [
    {
      "FUNC": [
        "name"
      ]
    }
  ],
  "bytesrange": [
    [ 68, 118 ]
  ],
  "linesrange": [
    [ [ 5, 4 ], [ 7, 5 ] ]
  ]
}
```

05

SWC



SWC REGISTRY

SWC Registry

Smart Contract Weakness Classification and Test Cases

The following table contains an overview of the SWC registry. Each row consists of an SWC identifier (ID), weakness title, CWE parent and list of related code samples. The links in the ID and Test Cases columns link to the respective SWC definition. Links in the Relationships column link to the CWE Base or Class type.

ID	Title	Relationships	Test cases
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	• odd_even.sol • odd_even_fixed.sol
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	• deposit_box.sol • deposit_box_fixed.sol • wallet.sol • wallet_fixed.sol
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	• hardcoded_gas_limits.sol
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	• access_control.sol • access_control_fixed_1.sol • access_control_fixed_2.sol
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	• Lockdrop.sol
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	• unused_state_variables.sol • unused_state_variables_fixed.sol • unused_variables.sol • unused_variables_fixed.sol

SOLGREP EXAMPLES

SWC-100

```
patterns:
  - pattern: |
      function $NAME( ... ) ... {
        ...
      }
  - not: |
      function $NAME( ... ) $VISIBILITY {
        ...
      }
```

SWC-101

```
patterns:
  - pattern: ... -= ...
  - pattern: ... += ...
  - pattern: ... *= ...
  - pattern: ... <<= ...
  - pattern: ... + ...
  - pattern: ... - ...
  - pattern: ... * ...
  - pattern: $_++
  - pattern: ++$_
    and:
     - pattern-root: pragma solidity $VERSION
# This will match all solidity  version, including <0.8.0
metavars-regex:
  $VERSION: (\d\.[0-7]\.\d*|<0\.8\.0)
```

06

CONCLUSIONS

- SATISFIED ON THE TOOL CAPABILITIES
- WRITE MORE SWC RULES
- EXTEND THE CODE TO SUPPORT CALL-TRACE AWARE QUERIES
- EXPECT THE SOLIDITY COMMUNITY TO CONTRIBUTE AND USE IT

https://github.com/fr0zn/solgrep