FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Analytics Platform for Ethereum Smart Contracts

Author:

Sarah Maria Hyatt

student number: zfj900

zfj900@alumni.ku.dk

Supervisor:

Boris Düdder

boris.d@di.ku.dk

Project in Practice

Department of Computer Science

University of Copenhagen

October 29, 2018

Contents

1 Introduction

The birth of Bitcoin in 2009 (Nakamoto, 2009) has led to a new era of decentralized cryptocurrencies, causing a public interest in electronic cash and decentralized systems, that do not rely on trusted parties (Luu, Chu, Olickel, Saxena, & Hobor, 2016). A blockchain, also known as a distributed ledger, is a private or public decentralized, append-only data structure, containing nodes that share a global state. A block contains multiple transactions and the nodes on the blockchain agree on an ordered set of blocks. A blockchain can therefore be viewed as a log of ordered transactions (Dinh et al., 2017).

Ethereum (Wood, 2014) is one of the emerging blockchain systems having their own cryptocurrency called Ether. Ethereum extends Bitcoin's design by creating its own Turing-complete language, Solidity (*Solidity*, n.d.), used to create smart contracts.

"A smart contract refers to the computation executed when a transaction is performed" (Dinh et al., 2017). Generally speaking a contract is a set of functions that encode and enforce agreements (Szabo, 1997). However, a transaction can only be executed if approved through consensus protocol based proof-of-work puzzles, among a large network of peers, called miners (Nicola Atzei & Cimoli, 2017). Each transaction is stored on the Ethereum blockchain. The execution of Ethereum smart contracts are Turing-complete programs in the form of Ethereum Virtual Machine (EVM) bytecode (Bhargavan et al., 2016).

The nature of a Turing-complete language provides rich computational power, but also makes software bugs inevitable (Dinh et al., 2017). Several studies suggest that one of the main causes of security breaches in Ethereum smart contracts are related to Solidity (Nicola Atzei & Cimoli, 2017), (Pettersson & Edström, 2016). Ethereum's market capitalisation is currently more than 2[1] billion dollars(*Ethereum Market Cap*, n.d.), making security a high risk for the numerous people involved. Many cases of exploitation has been discovered. One instance was in 2016 where a vulnerability was discovered (*Ethereum Blog*, n.d.). This DAO attack caused approximately 60 million dollars worth of assets stolen, due to 8000 Ethereum contracts containing security bugs (Dinh et al., 2017). Security in Ether contracts is therefore absolutely crucial.

The intention of this project is to create a platform that makes it possible to examine Ether smart contracts currently on the Ethereum blockchain. Leading to future prospects of researching patterns occurring in the mining process of Solidity contracts and to detect vulnerabilities. This means that we need to create a program that is able to collect and store Ether smart contracts. To begin with, we will collect contracts from EtherScan, because *"EtherScan is a Block Explorer, Search, API and Analytics Platform for Ethereum"* (*EtherScan*, n.d.), providing the most recent 5000 transaction attempts.

The above stated goals were partly achieved by creating a parser that works in two steps:

1. Collecting Ether smart contracts from EtherScan, inserting them into a database and generating abstract syntax trees (ASTs) for each contract.
2. All unique ASTs are added to a different table, which also include the number of copies of the identical contracts etc..

This project was done as a seven-week undergraduate project made in autumn 2018, at the Department of Computer Science, University of Copenhagen under the supervision of Boris Düdder.

[1]Retrieved: 24/10/2018 12:00 PM

2 Background

Automated gathering of data from the Internet is commonly known as *screen scraping, data mining, web-scraping* or *crawling*. This report refers to this process as crawling. Crawling was a tool for this project, as smart contracts from EtherScan were collected.

Application programming interfaces (APIs) provide a convenient stream of data and a standardised syntax used for two pieces of software to communicate (Mitchell, 2018), making it a useful tool for crawlers. As mentioned in the introduction, EtherScan is an Analytics Platform for Ethereum, providing an API that makes our crawling process simpler. However, EtherScan has a request limit of five per second (*EtherScan API*, n.d.), which means that the crawler should be constructed such that it does not exceed this limit. If the limit is exceeded we could experience bandwidth throttling (*Bandwidth Throttling*, n.d.), which will restrain our access to EtherScan.

In addition, EtherScan's API gives us access to only 5000 contracts, meaning that EtherScan is not a tool to collect the entire Ethereum blockchain.

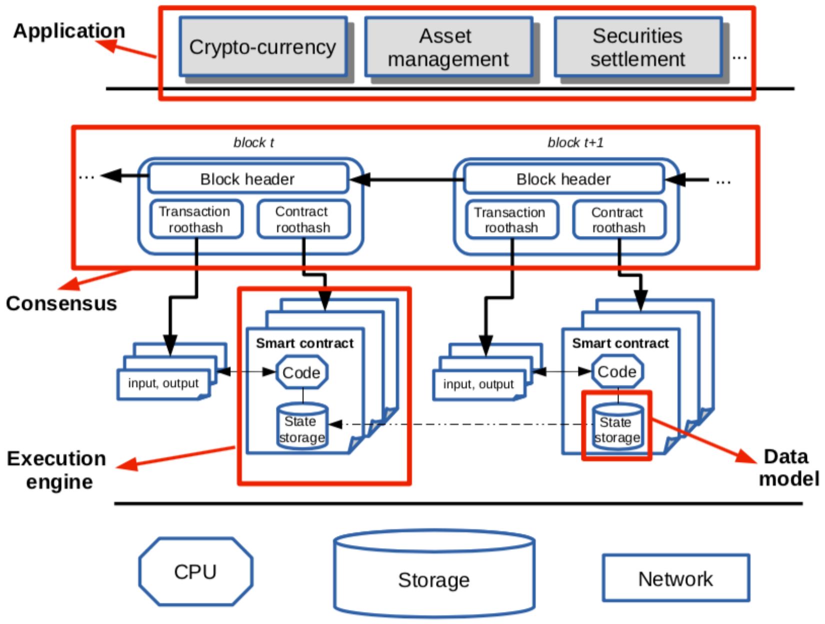

Fig. 1: Ethereum blockchain software stack (Dinh et al., 2017).

Figure 1 is an illustration of the logical components of the Ethereum blockchain software stack. The blockchain contains nodes sharing a global state (Dinh et al., 2017). The order of transactions determines the state of each contract, thus controlling the balance of each user (Nicola Atzei & Cimoli, 2017). However, it is the miners who decide the order of transactions in the block they mine (Luu et al., 2016). Figure 1 shows a fully validating node, where it takes Ethereum 12 seconds to find a new block and add it to the chain.

There are several known methods to breach security of this system. *Transaction Ordering Dependence* (TOD) can occur if the miner manipulates the final state, by reordering transactions in the block. This assumes that there are > 1 transactions in the block (Luu et al., 2016).

Another known attack *Reentrancy Vulnerability*, famous as the DAO Hack (*The DAO Hack*, n.d.), creates an infinite loop, because the balance to be withdrawn is not set to zero after having been withdrawn. The balance can then be withdrawn until the account is empty (Luu et al., 2016).

However, this is only two of many known attacks. In this report we present an initial platform that can research several patterns and occurrences of known as well as unknown attacks and vulnerabilities. In addition to learning the severity of code copying.

The plan for the solution was to:

1. Create a crawler that efficiently scrapes Ether smart contracts currently on the Ethereum blockchain.
2. Store the contracts such that they are easily accessible, while retaining necessary data for analytical purposes.

The combination of an MySQL database and ASTs was chosen to meet these goals. Using AST for pattern matching code is a known approach (Iulian Neamtiu & Hicks, 2005), in addition to program analysis for clone detection (Ira D. Baxter, Sant'Anna, & Bier, 1998) or error detection etc.. *"Some use abstract syntax that retain all of the structure of the concrete syntax trees plus additional positioning information used for error-reporting"* (Ægidius Mogensen, 2017). The AST will divide the contracts into a standardised structure, making it possible to detect the occurrences of clones in both a full tree and also sub-trees (Ira D. Baxter et al., 1998). This can be done by using tree-pattern matching, which, due to the tree-like structure, has efficient algorithms that can be used (Krebber, Barthels, & Bientinesi, 2017).

The demands for our program are as follows:

- The ability to continuously collect new contracts from the blockchain.
- Remove comments before storing them in the database to avoid taking up unnecessary space.
- Create an AST of each contract.
- Get the size of the contract.
- Detect whether contract copying appears on the chain.
- Store unique ASTs to avoid unnecessary future pattern matching on the same AST.
- Store the date and time the contract was collected for the purpose of debugging and/or analysing time specific dominant miners.

3 Analysis

One of the requirements is to store data, which makes a database a natural component. Our database contains two tables, referred to as *Contracts* and *Selected*. *Contracts* contains all the contracts, ASTs etc. and *Selected* contains unique ASTs without duplicates. The benefit of this solution is that *Selected* can be used to query unique ASTs, saving future computations. The disadvantage is that it takes up more storage by storing some values twice. An alternative solution would be to have one table and count the number of identical copies when needed. However, this would not give the benefits that the implemented solution does.

3.1 *Contracts*

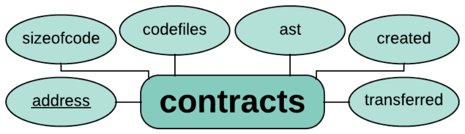

Fig. 2: Entity diagram of *contracts*.

The primary key, *address* in figure 2, is the first item to be collected by the crawler, as it works as a unique identifier for the contract. EtherScan includes the address in the url for the contract (*EtherScan Address Chart*, n.d.), which can be used as a tool when collecting contracts.

Attributes *codefiles* and *ast*, are stored as longblob data types. *codefiles* contains the contract collected by the crawler, and ast represents the AST created using the contract. *ast* contains the AST created using the contract. This means that while both Contracts and Selected store ASTs, Contracts contains many duplicate identical objects and Selected only unique copies.

In the previous section we stated that the user needs to be able to run the program at arbitrary points in time, and also in a state where the database has some data from previous runs. This means that we need an attribute that marks if the contracts have been transferred or not, hence the attribute *transferred*. *transferred* should be zero by default and set to 1 when the copy has been transferred to Selected. A benefit of this is that it is computationally faster to only look at the recently added items.

The indexed attribute *created* has two purposes.

1. Debugging, to see the behaviour and data input from a specific time of executing the crawler.
2. To detect certain time-specific behaviour that might occur among copies of added contracts.

The attribute *sizeofcode* computes and stores the byte size of a contract after its comments have been removed.

3.2 *Selected*

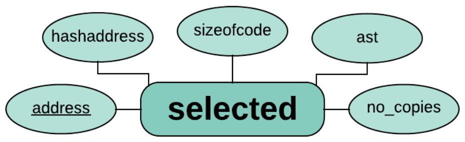

Fig. 3: Entity diagram of *selected*.

Attributes *address*, *sizeofcode* and *ast* in figure 3 are the same as described in the previous subsection. Section 3.4 will discuss this design.

One of our goals was to detect whether copying of contracts occurs. One of the reasons of creating ASTs was that an AST has a tree structure which makes it possible to do tree pattern matching on them. To test whether matching components are completely identical or not, the optimal solution would be tree pattern matching with a hashing algorithm. However, due to limited time, this program uses a work-around by only computing one hash value per AST, rather than for each sub-tree. This solution detects identical ASTs, but only if they are completely identical, sub-tree clones can not be detected.

The attribute *hashaddress* represents the hash value for each identical AST. It has to be saved to Selected because subsequent runs could scan identical contracts with the same AST. We have observed several cases of multiple different ASTs with the exact same size.

no_ copies has an integer which refers to the number of times an identical AST has occurred.

3.3 Putting it all Together

3.3.1 Gathering the Data:

The crawler does the following:

1. Communicates with EtherScan and fetches a list of addresses
2. Goes trough the list while:
   - 2.1 Communicating with *Contracts* to check if the address is stored or not
   - 2.2 If it already exists: the process continues
   - 2.3 If it does not exist:
     - 2.3.1 The crawler should use EtherScans API to get the contracts from the given address
     - 2.3.2 This contract is then processed by:
       - 2.3.2.1 Having its comments removed
       - 2.3.2.2 AST and size computed from the contract
       - 2.3.2.3 Address, contract, AST, size, the transferred mark and a time stamp is stored in *Contracts*

3.3.2 Sorting the Data:

After the data has been fetched, the application sorts in these steps:

1. Communicates with *Contracts* and fetches a list of all new items
2. Computes hash values of the ASTs

3. Groups the sizes with hash values
4. Counts the occurrences of the same hash values
5. Groups the data together, prepared for insertion
6. Looks up the hash value in *Selected*:
    6.1 If the hash values exists: the number of copies is updated
    6.2 If the hash values does not exist: the prepared data is inserted

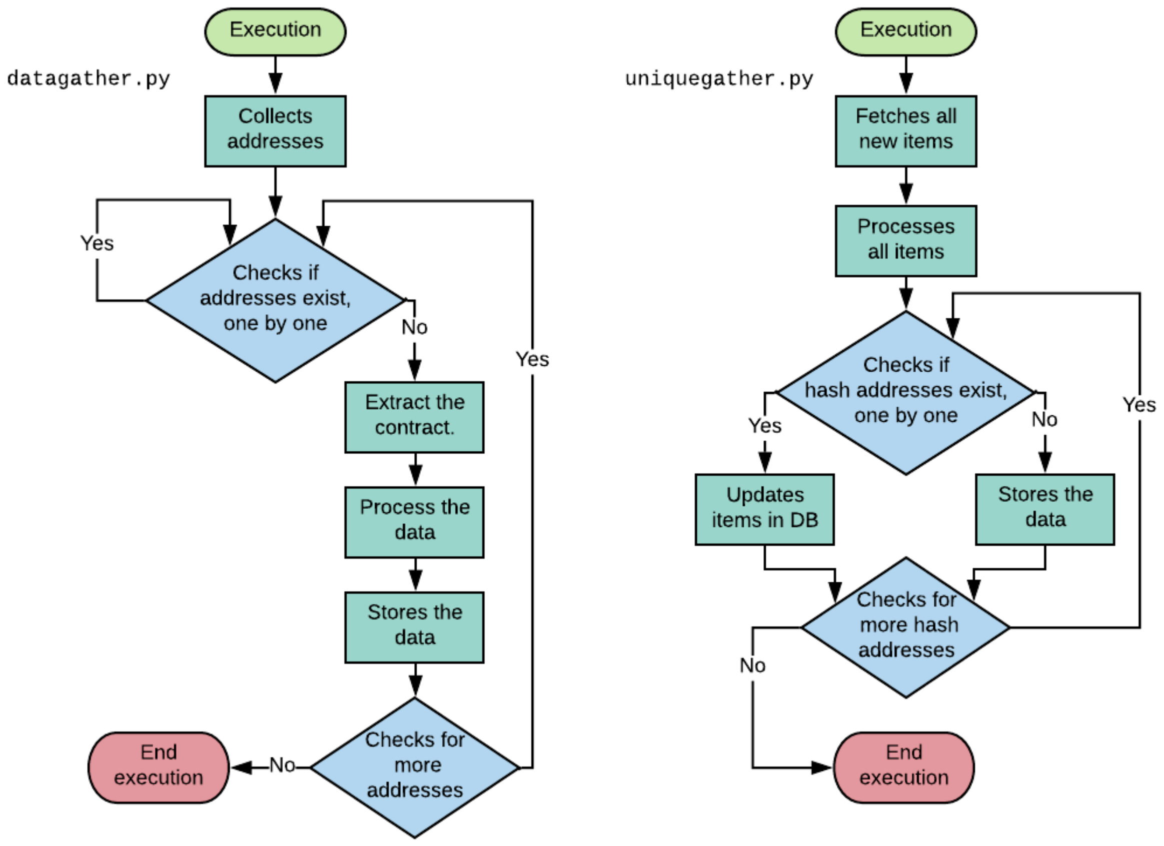

Fig. 4: The left flowchart illustrates the process in 3.3.1 the right flowchart illustrates the process in 3.3.2

4 Design

The component diagram in Figure 5 gives a full overview of the relations in the implementation. The application is generally tightly coupled due to the fact that the components both provide and receive data.

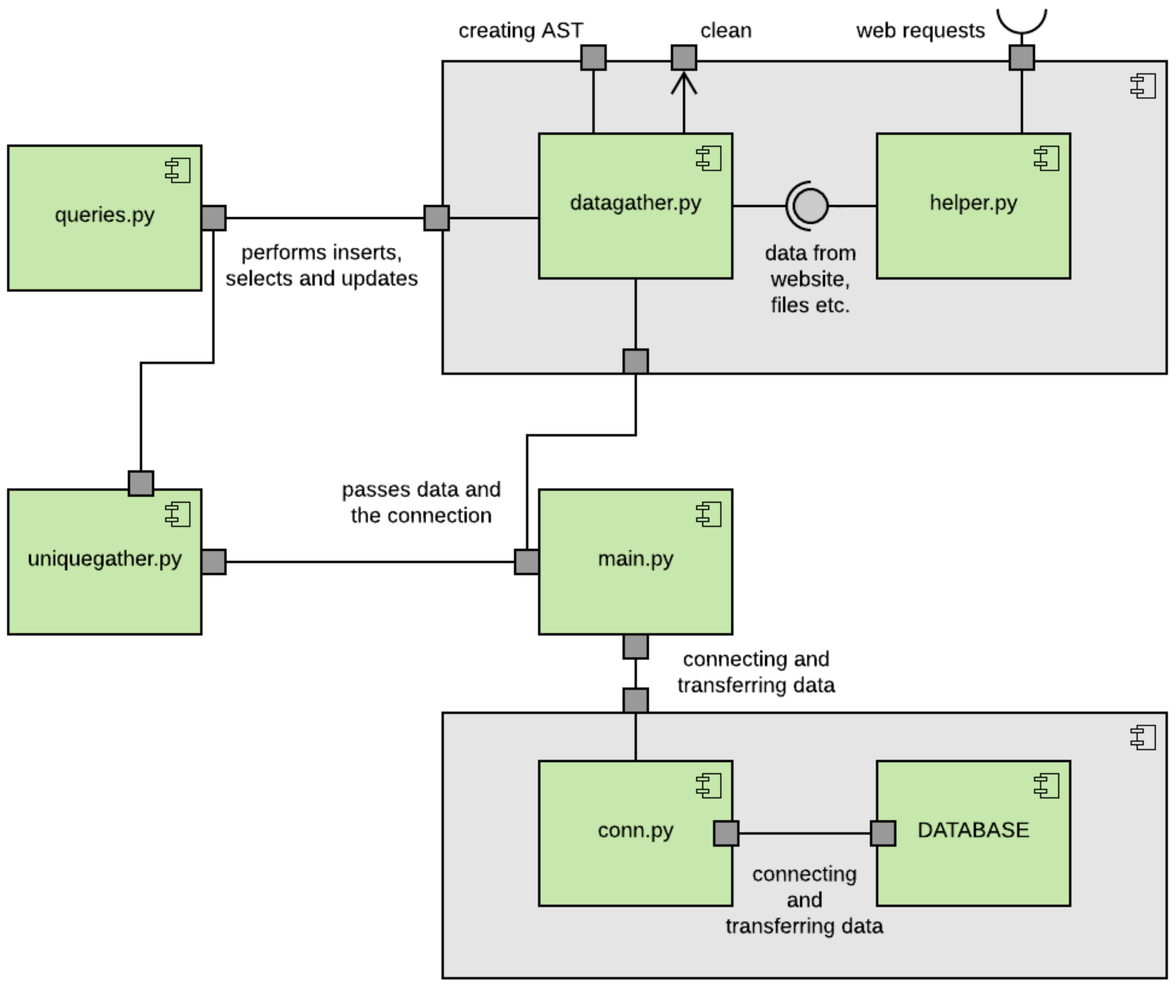
creating AST
clean
web requests
queries.py
performs inserts, selects and updates
datagather.py
data from website, files etc.
helper.py
passes data and the connection
uniquegather.py
main.py
connecting and transferring data
conn.py
DATABASE
connecting and transferring data

Fig. 5: Component Diagram of the Application.

`helper.py` retrieves information from EtherScan and passes it to `datagather.py`. `datagather.py` calls to system commands to create data and processes it with the help of methods and queries from `helper.py` and `queries.py`. `uniquegather.py` uses queries from `queries.py` to send data to and fetch data from the database. `main.py` owns the connection to the database and passes the cursor object with a live database connection to the other components.

This establishes a tight coupling but it also makes the program easy to run. Although there is a tight coupling to `main.py`, there is no coupling between `uniquegather.py` and `datagather.py`. `uniquegather.py` can only process data that is created if `datagather.py` is executed and new contracts are found, but they act as two separate components. `conn.py`, `helper.py` and `queries.py` are simply components that help do the job for `uniquegather.py` and `datagather.py` respectively.

5 User's Guide

5.1 Prerequisites and Requisites

Prerequisites to run this program are:

1 It has to be run on a Linux or Mac computer.

2 Python 3, including `pip`, `pymysql`, `requests`, `BeautifulSoup` 4, `numpy` and `matplotlib`.
3 MySQL, including MySQL server and preferably a administration tool such as PhPMyAdmin.
4 ANTLR4 will be installed if not installed already.

5.2 Running the Programs

5.2.1 First Step

`conn.py` contains the information specific setting up the connection to a MySQL server. This should be modified to suit the computer executing the application. The program nor tests will work unless this is set up correctly.

5.2.2 Creating the Database and Tables

The first step is to create the database. This is only to be done once. To create the database run the command: `python3 createdatabase.py`.

5.2.3 Collecting and Organizing Contracts

EtherScan has a limit of 5000 contracts, that can be accessed through their API, divided into 100 pages. Run the command `python3 main.py 1 100`, and you will start collecting valid contracts, that are not already stored in your database, from pages 1 to 100. This would, however, take a long time to execute and therefore it is adviced that this is done in chunks. This command can be run as many times as you like, since EtherScan is constantly updated with new smart contracts.

5.2.4 Running Tests

To run all the tests at once, go to the `tests` directory and use the command: `python3 -m unittest -v`. To run a test file separately, simply run the filename in Python 3, as this example: `python3 filename.py`, remaining in the `tests` directory.

6 Technical Description of the Program

This section will dive into some of the low level details of the implementation. As described in the Analysis and Design sections, the full program is a construction of multiple parts that work together.

6.1 Collecting Contracts, Creating AST's and storing it in the Database, `datagather.py`

6.1.1 Accessing a Page and Getting a BeautifulSoup Object

The `helper.py` module has two methods that create a request to a website; `getBsObject` and `getPage`. They Python `requests`(*Python Request Library*, n.d.) library, which allows sending of HTTP/1.1 requests. Additionally they use headers, which consists of a specified user-agent and accepts preferably HTML, XHTML, XML or any format available. This is stated as:

```
""  headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:62.0) Gecko
                        /20100101 Firefox/62.0 Chrome/39.0.2171.95 Safari/537.36",
            "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"}
```

When receiving objects over a network it is beneficial to specify headers. Agents maintain a description of their own processing state and the state of the world around them, making them ideal for crawlers. (Mitchell, 2018)

`getPage` returns a page session of the url using the specified headers, as shown below.

```
""page = session.get(url, headers=headers)
```

`getBsObject` returns a BeautifulSoup object from the requested page from above.

```
""  bs = BeautifulSoup(page.text, "html.parser")
```

`getBsObject` is presented in Appendix 9.4.

6.1.2 Gathering Contract Addresses

Appendix 9.1.1 provides the code for `getAddresses`, the method used to collect a list of addresses from EtherScan's section called Contract Internal Transactions. Only contracts without errors are in the chain. One run fetches a user specified number of pages between 1 and 100, each page containing 50 addresses. Essentially, this method is our crawler (Mitchell, 2018, p. 33-66). The first part of the method uses a helping function `getBsObject` (see Appendix 9.4) from `helper.py`, which opens each page in Contract Internal Transactions.

The second part of the method uses a css selector:

```
""  bs.select('tr > td:nth-of-type(4) a')
    for link in bs.select('tr > td:nth-of-type(5) a'):
      addresses.append(link.attrs['href'].split('/')[-1])
```

The following steps explain how it works:

1. finds a row `(tr)` that has a `(td)` element directly under it,
2. extracts the fifth `(td)` element `(td:nth-of-type(4))`,
3. extracts the link `(a)`
   e.g. `http://etherscan.io/address/0x006bc3dc8e84930a53b022b0b3cae89e34bc5eea`
4. extracts the address from of the link, and appends it to the list.

Once we have all the addresses we can run them through the method `getContracts`, (see Appendix 9.1.2),

6.1.3 MySQL Queries for Inserts, Updates and Selecting

Creating a connection to the database is done by importing the module `conn.py`. This uses the Python database connector library `pymysql`. The `main.py` maintains a connection through the entire execution of the program. This works because a cursor object is created and passed to each method that uses it.

All queries are collected into a single module called `queries.py`. An example of such a query is the following method `updateNoCopies`.

```
1  ""def updateNoCopies(cur, new_no_copies, hashes):
2    try:
3      update_qry = """UPDATE smartcontract_db.selected
4                      SET no_copies=("%s") WHERE hashaddress=("%s")"""
5      update_data = (new_no_copies, hashes)
6      cur.execute(update_qry, update_data)
7      cur.connection.commit()
8    except:
9      print("Unexpected error updating the number of copies:", sys.exc_info()[0])
```

This method's function is to update the number of copies registered in *Selected*. `updateNoCopies` takes three arguments `cur, new_no_copies` and `hashes`. `cur` is the cursor object for the current connection. `new_no_copies` is the new number of copies that need to be updated. `hashes` is the hash address whose number of copies needs updating. Each query handles potential exceptions and provides an error message that shows what the problem is and which method is troubling. Lines 2 and 8-9 show this consideration.

Lines 3-4 is the actual MySQL update query which matches with the hash address given in line 5 and updates the number of copies with our new value. Lines 6-7 execute and commit the query.

6.1.4 Removing Comments

Solidity uses C-like comments such as; `//` and `/* */`. The comments can occur on one or multiple lines. The `removeComments` method below applies Python's regex (*Python Regex Library*, n.d.) library to extract all comments from the contracts.

```
1  ""def removeComments(contract):
2    replacement = " "
3    comments = re.compile(r'(?<!https:)//.*?$|/\*.*?\*/',
4      re.DOTALL | re.MULTILINE
5    )
6    return re.sub(comments, replacement, contract)
```

The negative look behind on line 3 `(?<!https:)` ensures purposely added urls are not removed. `//.*?$` removes everything that comes after `//`, including `//`. `.*?/` removes `/* */` comments. `re.DOTALL` makes '.' represent all characters. `re.MULTILINE` matches subsequent lines in multi-line comments.

In line 6 all comments are replaced by a single space, hence contracts with a high number of comments will contain a lot of white space after their removal.

6.1.5 Creating Abstract Syntax Trees

The ASTs are created by saving the contracts to a file and then running the file through the shell script `create_ast.py`, using Python's `subprocess` library (*Python Subprocess Library*, n.d.). The reason for this workaround is that `create_ast.py` uses Another Tool For Language Recognition (*ANTLR*, n.d.) (ANTLR). ANTLR is a parser generator that uses LL parsing, which parses input from left to right, performing leftmost derivation of the sentence (*LL Parsing*, n.d.). ANTLR4 provides a Solidity grammar (*ANTLR Solidity Grammar*, n.d.) in a file called `Solidity.g4`. This is imported to the script and lexer, parser and additional parsing tools are created. If ANTLR4 is not installed the script will install it. The script is given in Appendix 9.6 and an example of a saved AST in Appendix 9.9, as well as a visualisation of a AST in Appendix 9.7.

6.1.6 Collecting Contracts

The previous mentioned method `getAddresses` returns a list of addresses pulled from the pages defined when running the application. `getContracts` overall performs as described in section 3.3.1. Below is a short section of the method, which shows how contracts are extracted.

```
1  ""page = helper.getPage('https://api.etherscan.io/api?module=contract&action=getsourcecode&address
     ={}&apikey=RMo8wU2K53Mm'.format(addr))
2
3      res = page.json()['result'][0]
4      if not isinstance(res, str):
5        source = res['SourceCode']
6
7        if len(source) > 0:
```

Line 1 opens EtherScan's API at the page related to "get sourcecode" and the address specified in the url. Their API responds with JSON (*JSON*, n.d.) data. The contract is saved in an array with one element called `'result'`, line 3 above extracts this content using a built-in Python function `json()`. Below is a demonstration of a short snippet of what `res` contains in the case of an address containing a contract.

```
{"status":"1","message":"OK","result":[{"SourceCode":"pragma solidity ^0.4.14;
```

Line 4 assures the content is not a string, in which case we would be dealing with an address with no contract. Line 5 extracts the content of the object in `res` called `'SourceCode'` and this is the contract which we save as `source`. Lastly, line 7 confirms that there in fact is a contract, before executing the rest of the method as described in section 3.3.1.

6.2 Organising Unique Contracts to a Separate Table, `uniquegather.py`

Section 3.3.2 describes the overall flow of how `uniquegather.py` works.

6.2.1 Computing a hash value of each AST

`computeHash` is included in the module `uniquegather.py`. Python's library `hashlib`(*Hashlib Docs.*, n.d.) uses secure hash algorithms (SHA) and message-digest algorithms such as MD5. It detects identical objects as they would have the same hash value.

```
1 ""def computeHash(ast):
2     hash_obj = hashlib.md5(ast.encode())
3     return hash_obj.hexdigest()
```

`computeHash` takes an AST as an argument and runs it through the MD5 algorithm (line 2). The *hashaddress* attribute is returned as a string object containing only hexadecimal digits.

6.2.2 Counting the Copies

The method `sortData` uses the query `selectNonTransferred`[2] from `queries.py` to extract a list `datalist` including `address, sizeofcode` and `ast` from *Contracts*.

We loop through `datalist` twice, the first time we creating hash values of the ASTs and group each matching size to the hash value, in the dictionary `sizes`. Before running through `datalist` the second time we count the occurrences of sizes for each hash value and store it in the dictionary `nocopies`. It is shown in the code below.

```
""# grouping the sizes with the matching hash value
  for elm in datalist:
    size_before = list(elm.values())[1]
    ast_before  = list(elm.values())[2]
    hashkey_before = helper.computeHash(str(ast_before))
    sizes.setdefault(hashkey_before, []).append(size_before)

  # adding the total number copies to each hash in dict nocopies
  for x in sizes:
    nocopies.update({x:(len(sizes[x]))})
```

The second time we loop through `data_list` we wish to create a nested list, where each inner list contains all the data necessary for inserting to *Selected*. The hash value of an AST is computed again and then we perform a lookup into the dictionary `nocopies`, to find the matching number of copies. When this is done, the data is inserted to our nested list `collecteddata`. The code is displayed below.

```
""# combining all the matching values in a list of lists
  for elm in datalist:
    addr, size, ast = (list(elm.values())[i] for i in range(3))
    hashkey = helper.computeHash(str(ast))
    # looking up the number of copies matching the hash value
    copies  = nocopies.get(hashkey, 'The hash value is not found')

    if not hashkey in [j for i in collecteddata for j in i]:
      insrtions = [hashkey, addr, size, copies, ast]
      collecteddata.append(list(insrtions))

  return collecteddata
```

6.2.3 Inserting or Updating *Selected*

The important part of `transferUnique`[3] is that it looks up the hash value from `collecteddata` and then uses the query `selectCopiesFromHash` from `queries.py` to check if the hash value is already stored (line 6). Lines 9-10 below use an insert query from `queries.py` if the hash does not already exist, while lines 16-18 get the number of copies already stored. Lines 21-23 update the matching hash value from *Selected* with the updated number of copies. Lastly at line 26, the query `updateTransferred` from `queries.py` sets the transfer attribute of all rows *transferred* to 1.

```
1  ""  # runs through the list  an either inserts or updates to Selected
2    for insrtions in collecteddata:
3      hashes, addrss, sizes, copies, asts = (insrtions[i] for i in range(5))
4
5      # gets the matching hashvalues from the db, if any
6      hashaddresses = queries.selectCopiesFromHash(cur, hashes)
7
8      # inserts the unique AST
9      if len(hashaddresses) == 0:
10       queries.insertToSelected(cur, addrss, hashes, sizes, asts, copies)
```

[2]See Appendix 9.5
[3]See Appendix 9.3

```

    # each iteration restores the number of copies to zero
    nocopies_in_db = 0

    # gets the number of copies already stored
    for elem in hashaddresses:
      hashkey_in_db, nocopies_in_db = (list(elem.values())[i] for i in range(2))
      oldcopies.setdefault(hashkey_in_db, []).append(nocopies_in_db)

    # updates the stored hash with the new number of copies
    if len(hashaddresses) == 1:
      new_no_copies = copies+nocopies_in_db
      queries.updateNoCopies(cur, new_no_copies, hashes)

  # updates: contracts.transferred, to avoid duplicate inputs
  queries.updateTransferred(cur)
```

7 Evaluation

7.0.1 Set-up

This project was performed using a MacBook Air from 2015, with a 1,6 GHz Intel Core i5 processor and 4 GB RAM. The wireless Internet connection had an average download speed of 80 Mbps[4].

7.0.2 Performance

Collecting and sorting 3009 contracts to an initially empty database took 6 hours, 54 minutes and 14 seconds, for an average of 8 seconds per contract. The crawler examined 15242 addresses and inserted 3009 of them. This means that approximately 20% of the addresses were valid contracts not already stored in the database.

The average number of requests per second was 1,33. Given more time, it would be interesting to look at the possibilities of implementing multi threading, to increase the performance.

7.0.3 Results

In a data collection sample of 3009 contracts there were 220 unique contracts, meaning that approximately 93% were identical contracts that occur more than once. Figure 5 below demonstrates the distribution of the number of identical copies occurring based on sizes.

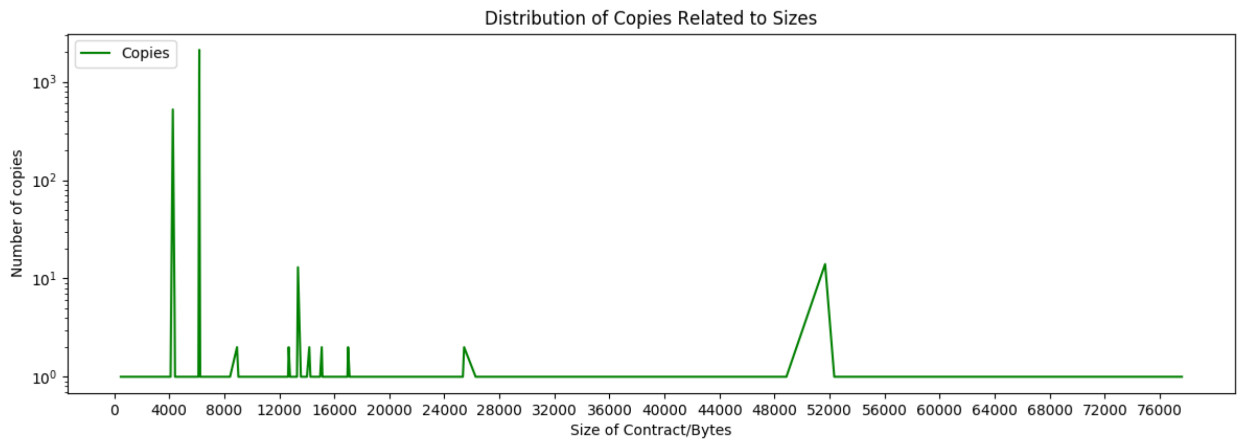

Fig. 6: The number of identical copies according to size.

The size range in this collection is between 481 to 77605 bytes per contract. The highest three peaks on figure 5 are:

[4]measured on https://www.speedtest.net/

1. The last and third largest peak has the size 51675 with 14 copies. Looking into it the producer of the contract is a company[5] that offers people to freely use the contracts they make. However, this contract shows warnings of low-high severity Solidity compiler Bugs.

2. The second largest peak has the size 4265 with 523 copies. This contract is created by a cryptocurrency exchange[6]. This contract too shows warnings of low-high severity Solidity compiler Bugs.

3. The highest peak is in fact two peaks: size 6204 with 122 copies and size 6190 with 2112 copies. Size 6190 with 2112 copies is particularly peculiar. It only started showing up after October 9, 2018 and has since been the most dominant of all transactions, resulting in %70 of transactions. All the transactions I have looked into have 0 Ether, $0 Ether value and 0 transactions. In addition, all the contracts self-destruct (*Solidity Selfdestruct*, n.d.) immediately after execution. Our crawler, however, does manage to get the contract. The owner of this contract is not currently identified. Size 6204 with 122 copies also shows warnings of low-high severity Solidity compiler Bugs. However, further analysis is necessary to identify these two peaks.

7.1 Code Flaws

Given more time the subjects below should be considered:

7.1.1 Redesigning

Figure 4 in section 4 provides a overview of the relations between the components in the application. Most components are tightly coupled and have many two-way dependencies. This should be refactored into relations that are less dependent and less tightly coupled together.

7.1.2 Implementing Tree-Pattern Matching for Detecting Copied Contracts

As mentioned in Analysis, combining tree-pattern matching with hashing algorithms to detect clones would be a better solution rather than computing hash values of the full ASTs. Appendix 9.9 shows an example of two ASTs added that only differ by a few numbers in names and Solidity version. However, they result into two different hash values and using the tree-pattern matching method could have detected this.

7.1.3 Cleaner Code

`uniquegather.py` and `datagather.py` have code and design smells (Martin & Martin, 2007). They have methods that are too long and methods which also creates rigidity and opacity. `uniquegather.py` has methods containing somewhat duplicated code and long parameter lists.

7.1.4 Reducing Unnecessary Consumption of Space

The solution stores ASTs of the contracts in two tables, which is unnecessary consumption of space. A better solution would be to create the ASTs while extracting contracts to *Selected*.

7.1.5 Relational Database Design

The current solution uses one database with two tables that have some of the same attributes. Performing normalization could result in a better solution with relations instead of duplicates.

7.1.6 More Tests

The current testing environment is not complete. More tests need to be implemented to assure the accuracy of the program.

[5]http://www.oraclize.it/
[6]https://bittrex.com/

7.2 Testing

The application uses Unit testing. To recreate the test results follow the instructions in section 5. Unfortunately the testing is incomplete, tests for `queries.py` are missing and tests in `uniquegather.py` are currently scarce. A short overview of the test implementations:

- `helper.py`
  - `getBsObject`
    - Returns a BeautifulSoup object.
    - Handles request exceptions.
    - Returns the right content.
  - `getPage`
    - Returns a request object.
    - Handles request exceptions.
  - `clean`
    - The file is removed.
    - Does not crash if the file does not exist.
    - Handles getting a directory.
  - `readFile`
    - The file is read accurately.
    - Returns the correct type.
  - `saveToFile`
    - The saved input is intact.
  - `removeComments`
    - The output file has removed comments as it should have.
    - Lines of code are not removed.
    - Url addresses are not removed.
- `uniquegather.py`
  - `computeHash`
    - Returns the same hash value for the same file.
    - Returns a different hash value for two different files.
- `datagather.py`
  - `getAddresses`
    - The correct type is returned.
    - The addresses are strings of size 42.
  - `getContracts`
    - Addresses correctly inserted to Contracts.
    - Three columns are correctly gathered to Contracts.
    - Creates and inserts a AST accurately to Contracts.

8 Conclusion

Within the time limits, it was not feasible to create ASTs without using a shell script and a temporary file. Utilising ANTLR4 was a time saving way of creating ASTs, but the time needed to incorporate the creation of ASTs to the application was too time consuming for a seven week project. The workaround puts some restrictions on the program with regards to executable operating systems and time consumption.

In addition, having implemented tree-pattern matching for clone and errors detection would also have been preferable, but time did not permit this either.

Nevertheless, the project is a well working Ethereum smart contract collector. It uses 8 seconds on average to collect a valid contract identified by its unique address, remove its comments, gets its size, create an AST of it and store all information in *Contracts*. After that unique ASTs are computed using hash algorithms and inserted into *Selected*, along with the address, size, and the number of copies occurring of the identical AST. The application is easy to execute as the process is run through a main file which also handles the connection to the database. All database queries and methods that connect to external interfaces are handled with exceptions, making the program easier to debug.

Overall the goals stated in the introduction have been met, although adjustments and improvements are needed.

8.1 Future Work

8.1.1 Refactoring

Refactoring the poor design choices (section 7.1) and implementing multi threading (7.0.3) to increase performance would improve the software substantially.

8.1.2 Extending

Since EtherScan imposes limits on how many smart contracts we will be able to gain access to, it should be seen as a testing platform. It could be interesting to implement tree-pattern algorithms and test them against clone detection and known as well as unknown vulnerabilities.

The opportunity of expanding the collection by gathering contracts from other providers such as Google Big Query (*Google Big Query*, n.d.) is also an option. Once these algorithms are implemented and tested it would be interesting to extract the entire Ethereum blockchain and perform the same pattern matching.

Understanding the full amount of contract copying is interesting for several reasons. Copying contracts or even parts of contracts is considered plagiarism (*Wikipedia, Plagiarism*, n.d.). Ethereum claims to be a secure decentralised generalised transaction ledger(Wood, 2014), dealing with peoples finances. However, as mentioned throughout this report, studies show the opposite (Luu et al., 2016), (Kevin Delmolino, Miller, & Shi, 2015). In our findings we saw that 90% of the contracts extracted were in fact identical copies even containing low-high risk security warnings. Even so, they were still executed on the blockchain. It would be interesting to cluster the owners of each contract with > 1 appearances on the blockchain, to get an idea of whether the same miners reuse their own contract or others do.

9 Appendix

9.1 Methods from data_gather.py

9.1.1 getAddresses

Getting a list with addresses to potential smart contracts.

```
""  def getAddresses(p_from, p_to):
    addresses = []

    for i in range(p_from, p_to):
      bs = helper.getBsObject("""https://etherscan.io/txsInternal?ps=100&&valid=true&p=""" + str(i
    ))

      bs.select('tr > td:nth-of-type(4) a')
      for link in bs.select('tr > td:nth-of-type(5) a'):
        addresses.append(link.attrs['href'].split('/')[-1])

    return addresses
```

9.1.2 getContracts

Uses a list of addresses to

```
""def getContracts(cur, addresses):
  transferred = 0
  counter = 0

  for addr in addresses:
    # checks if it already is in the database
    if queries.selectAddrContracts(cur, addr) == 0:

      # inserts the address in etherscan.io's api and opens the page
      page = helper.getPage('https://api.etherscan.io/api?module=contract&action=getsourcecode&
    address={}&apikey=RMo8wU2K53Mm'.format(addr))

      # extracting the contract
      res = page.json()['result'][0]
      if not isinstance(res, str):
        source = res['SourceCode']

        # adds to the address to the set, if there is a contract
        if len(source) > 0:
          # removing the comments
          contract = helper.removeComments(source)

          # gets the code size
          codesize = len(contract)

          # create the AST via script
          helper.saveToFile(addr, contract)
          ast = helper.createAST(addr)

          # adds to database.
          if queries.insertToContracts(cur,addr,codesize,contract,ast,transferred):
            counter += 1
            helper.clean(addr)

        else:
          continue
    else:
      continue

  return counter
```

9.2 sortData from uniquegather.py

```
""def sortData(cur):
  # selecting the newly added items from contracts
  datalist = queries.selectNonTransferred(cur)

  collecteddata, insrtions = ([] for i in range(2))
  nocopies, sizes = ({} for i in range(2))

  # grouping the sizes with the matching hash value
  for elm in datalist:
    size_before = list(elm.values())[1]
    ast_before  = list(elm.values())[2]
    hashkey_before = computeHash(str(ast_before))
    sizes.setdefault(hashkey_before, []).append(size_before)

  # adding the total number copies to each hash in dict nocopies
  for x in d_sizes:
    nocopies.update({x:(len(sizes[x]))})

  # combining all the matching values in a list of lists
  for elm in datalist:
    addr, size, ast = (list(elm.values())[i] for i in range(3))
    hashkey = computeHash(str(ast))
    # looking up the number of copies matching the hash value
    copies  = nocopies.get(hashkey, 'The hash value is not found')

    if not hashkey in [j for i in collecteddata for j in i]:
      insrtions = [hashkey, addr, size, copies, ast]
      collecteddata.append(list(insrtions))

  return collecteddata
```

9.3 `transferUnique` from `uniquegather.py`

```
""def transferUnique(cur, collecteddata):
  oldcopies = {}

  # runs through the list  an either inserts or updates to Selected
  for insrtions in collecteddata:
    hashes, addrss, sizes, copies, asts = (insrtions[i] for i in range(5))

    # gets the matching hashvalues from the db, if any
    hashaddresses = queries.selectCopiesFromHash(cur, hashes)

    # inserts the unique AST
    if len(hashaddresses) == 0:
      queries.insertToSelected(cur, addrss, hashes, sizes, asts, copies)

    # each iteration restores the number of copies to zero
    nocopies_in_db = 0

    # gets the number of copies already stored
    for elem in hashaddresses:
      hashkey_in_db, nocopies_in_db = (list(elem.values())[i] for i in range(2))
      oldcopies.setdefault(hashkey_in_db, []).append(nocopies_in_db)

    # updates the stored hash with the new number of copies
    if len(hashaddresses) == 1:
      new_no_copies = copies+nocopies_in_db
      queries.updateNoCopies(cur, new_no_copies, hashes)

  # updates: contracts.transferred, to avoid duplicate inputs
  queries.updateTransferred(cur)
```

9.4 `getBsObject` from `helper.py`

Getting a Beautifulsoup object of a url. This method is highly inspired by (Mitchell, 2018, p. 219)

```
""def getBsObject(url):
  # creating a session and defining headers and output preferences
  session = requests.Session()
  headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:62.0) Gecko/20100101
    Firefox/62.0",
             "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"}
  # request the page given by url
  try:
    page = session.get(url, headers=headers)
  except requests.exceptions.RequestException:
    return None
  # creates the BeautifulSoup object
  bs = BeautifulSoup(page.text, "html.parser")

  return bs
```

9.5 selectNonTransferred from queries.py

```
""def selectNonTransferred(cur):
  try:
    select_qry = """SELECT address,sizeofcode,ast
                    FROM smartcontract_db.contracts
                    WHERE transferred = 0"""
    cur.execute(select_qry)
    cur.connection.commit()
  except:
      print("Unexpected error selecting transferred:", sys.exc_info()[0])

  return cur.fetchall()
```

9.6 Script for creating an AST

```
""#!/bin/bash

# defines needed variables.
ANTLR_JAR="antlr4.jar"
GRAMMAR="Solidity"
DLANGUAGE="-Dlanguage=Python3"
START_RULE="sourceUnit"
TYPE="-tree"
FILE=$1

# makes sure that antlr4 is installed.
if [ ! -e "$ANTLR_JAR" ]; then
  curl http://www.antlr.org/download/antlr-4.7-complete.jar -o "$ANTLR_JAR"
fi

# creates a directory for the parsing tools, -p is so it can be used as an operand.
mkdir -p parsetools/

# creates Solidity lexer, parser and listeners.
java -jar $ANTLR_JAR $GRAMMAR.g4 -o src/

# creates the Solidity parsing tools and adds them to parsetool.
javac -classpath $ANTLR_JAR src/*.java -d parsetools/

# creates the AST from the given contract file.
java -classpath $ANTLR_JAR:parsetools/ org.antlr.v4.gui.TestRig "$GRAMMAR" "$START_RULE" "$TYPE" <
     "$FILE" 2>&1
```

9.7 Demonstration of the Smallest AST with Size 481 Bytes

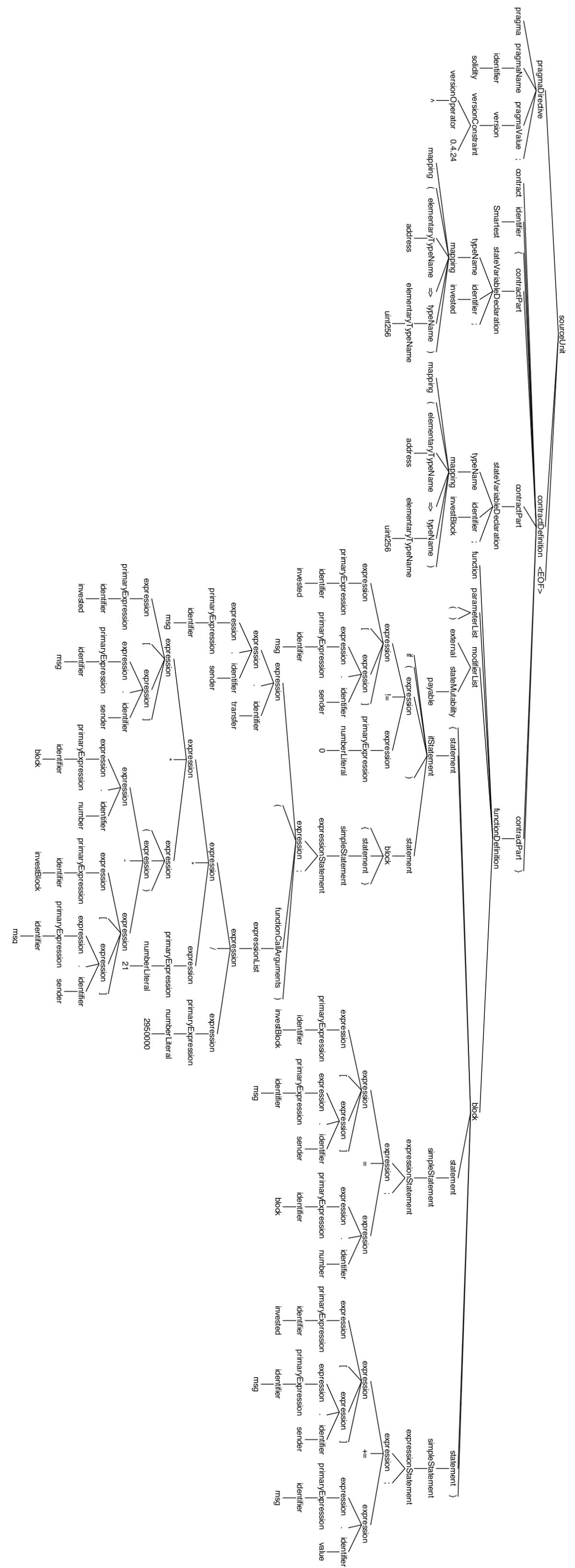

Fig. 7: A 481 byte big AST made with ANTLR4 using gui.

9.8 The Contract the Smallest AST with Size 481 Bytes

```
pragma solidity ^0.4.24;

contract Smartest {
    mapping (address => uint256) invested;
    mapping (address => uint256) investBlock;

    function () external payable {
        if (invested[msg.sender] != 0) {

            msg.sender.transfer(invested[msg.sender] * (block.number - investBlock[msg.sender])
            * 21 / 2950000);
        }

        investBlock[msg.sender] = block.number;
        invested[msg.sender] += msg.value;
    }
}
```

9.9 Pitfalls from Computing Hash Values of AST's

Below is two AST's with different addresses, that are identical besides the Solidity version and a number attached to a contract name.

AST of Address: 0x39a765ecf417f5eecbb58d6b0482f81f4d825fff:

```
(sourceUnit (pragmaDirective pragma (pragmaName (identifier solidity)) (
    pragmaValue (version (versionConstraint (versionOperator ^) 0.4.24))) ;) (
    contractDefinition contract (identifier EasyInvest3) { (contractPart (
    stateVariableDeclaration (typeName (mapping mapping ( (elementaryTypeName
    address) => (typeName (elementaryTypeName uint256)) ))) public (identifier
    invested) ;)) (contractPart (stateVariableDeclaration (typeName (mapping
    mapping ( (elementaryTypeName address) => (typeName (elementaryTypeName
    uint256)) ))) public (identifier atBlock) ;)) (contractPart (
    functionDefinition function (parameterList ( )) (modifierList external (
    stateMutability payable)) (block { (statement (ifStatement if ( (expression
     (expression (expression (primaryExpression (identifier invested))) [ (
    expression (expression (primaryExpression (identifier msg))) . (identifier
    sender)) ]) != (expression (primaryExpression (numberLiteral 0)))) ) (
    statement (block { (statement (simpleStatement (
    variableDeclarationStatement (variableDeclaration (typeName (
    elementaryTypeName uint256)) (identifier amount)) = (expression (expression
     (expression (expression (expression (expression (primaryExpression (
    identifier invested))) [ (expression (expression (primaryExpression (
    identifier msg))) . (identifier sender)) ]) * (expression (
    primaryExpression (numberLiteral 3)))) / (expression (primaryExpression (
    numberLiteral 100)))) * (expression ( (expression (expression (expression (
    primaryExpression (identifier block))) . (identifier number)) - (expression
     (expression (primaryExpression (identifier atBlock))) [ (expression (
    expression (primaryExpression (identifier msg))) . (identifier sender)) ]))
     ))) / (expression (primaryExpression (numberLiteral 5900)))) ;))) (
    statement (simpleStatement (expressionStatement (expression (expression (
    expression (expression (primaryExpression (identifier msg))) . (identifier
    sender)) . (identifier transfer)) ( (functionCallArguments (expressionList
    (expression (primaryExpression (identifier amount))))) )) ;))) })))) (
    statement (simpleStatement (expressionStatement (expression (expression (
    expression (primaryExpression (identifier atBlock))) [ (expression (
    expression (primaryExpression (identifier msg))) . (identifier sender)) ])
    = (expression (expression (primaryExpression (identifier block))) . (
    identifier number))) ;))) (statement (simpleStatement (expressionStatement
    (expression (expression (expression (primaryExpression (identifier invested
    ))) [ (expression (expression (primaryExpression (identifier msg))) . (
    identifier sender)) ]) += (expression (expression (primaryExpression (
    identifier msg))) . (identifier value))) ;))) }))) }) <EOF>)
```

AST of Address: 0x6519d6dfd11363cf0821809b919b55f794fe0cb5:

```
(sourceUnit (pragmaDirective pragma (pragmaName (identifier solidity)) (
    pragmaValue (version (versionConstraint (versionOperator ^) 0.4.25))) ;) (
    contractDefinition contract (identifier EasyInvest6) { (contractPart (
    stateVariableDeclaration (typeName (mapping mapping ( (elementaryTypeName
    address) => (typeName (elementaryTypeName uint256)) ))) public (identifier
    invested) ;)) (contractPart (stateVariableDeclaration (typeName (mapping
    mapping ( (elementaryTypeName address) => (typeName (elementaryTypeName
    uint256)) ))) public (identifier atBlock) ;)) (contractPart (
    functionDefinition function (parameterList ( )) (modifierList external (
    stateMutability payable)) (block { (statement (ifStatement if ( (expression
     (expression (expression (primaryExpression (identifier invested))) [ (
    expression (expression (primaryExpression (identifier msg))) . (identifier
    sender)) ]) != (expression (primaryExpression (numberLiteral 0)))) ) (
    statement (block { (statement (simpleStatement (
    variableDeclarationStatement (variableDeclaration (typeName (
    elementaryTypeName uint256)) (identifier amount)) = (expression (expression
     (expression (expression (expression (expression (primaryExpression (
    identifier invested))) [ (expression (expression (primaryExpression (
    identifier msg))) . (identifier sender)) ]) * (expression (
    primaryExpression (numberLiteral 3)))) / (expression (primaryExpression (
    numberLiteral 100)))) * (expression ( (expression (expression (expression (
    primaryExpression (identifier block))) . (identifier number)) - (expression
     (expression (primaryExpression (identifier atBlock))) [ (expression (
    expression (primaryExpression (identifier msg))) . (identifier sender)) ]))
     ))) / (expression (primaryExpression (numberLiteral 5900)))) ;))) (
    statement (simpleStatement (expressionStatement (expression (expression (
    expression (expression (primaryExpression (identifier msg))) . (identifier
    sender)) . (identifier transfer)) ( (functionCallArguments (expressionList
    (expression (primaryExpression (identifier amount))))) )) ;))) })))) (
    statement (simpleStatement (expressionStatement (expression (expression (
    expression (primaryExpression (identifier atBlock))) [ (expression (
    expression (primaryExpression (identifier msg))) . (identifier sender)) ])
    = (expression (expression (primaryExpression (identifier block))) . (
    identifier number))) ;))) (statement (simpleStatement (expressionStatement
    (expression (expression (expression (primaryExpression (identifier invested
    ))) [ (expression (expression (primaryExpression (identifier msg))) . (
    identifier sender)) ]) += (expression (expression (primaryExpression (
    identifier msg))) . (identifier value))) ;))) }))) }) <EOF>)
```

References

Antlr. (n.d.). Retrieved from http://www.antlr.org/

Antlr solidity grammar. (n.d.). Retrieved from https://github.com/solidityj/solidity-antlr4

Bandwidth throttling. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Bandwidth_throttling

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., ... Zanella-Beguelin, S. (2016). *Formal verification of smart contracts* (Tech. Rep.). Harvard University and Microsoft Research and Inria. doi: https://dl.acm.org/citation.cfm?id=2993611

The dao hack. (n.d.). Retrieved from https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413#code

Dinh, T. T. A., Liu, R., Zhang, M., Chen, G., Ooi, B. C., & Wang, J. (2017). *Untangling blockchain: A data processing view of blockchain systems* (Tech. Rep.). IEEE.

Ethereum blog. (n.d.). Retrieved from https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/

Ethereum market cap. (n.d.). Retrieved from https://coinmarketcap.com/currencies/ethereum/

Etherscan. (n.d.). Retrieved from https://etherscan.io/aboutus

Etherscan address chart. (n.d.). Retrieved from https://etherscan.io/chart/address

Etherscan api. (n.d.). Retrieved from https://etherscan.io/apis

Google big query. (n.d.). Retrieved from https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-how-we-built-dataset

Hashlib docs. (n.d.). Retrieved from https://docs.python.org/3/library/hashlib.html

Ira D. Baxter, L. M., Andrew Yahin, Sant'Anna, M., & Bier, L. (1998). *Clone detection using abstract syntax trees* (Tech. Rep.). Departamento de Informatica Pontificia Universidade Catolica, Eaton Corporation and Semantic Designs. doi: http://www.semanticdesigns.com/Company/Publications/ICSM98.pdf

Iulian Neamtiu, J. S. F., & Hicks, M. (2005). *Understanding source code evolution using abstract syntax tree matching* (Tech. Rep.). Department of Computer Science University of Maryland at College Park. doi: http://www.cs.umd.edu/ mwh/papers/evolution.pdf

Json. (n.d.). Retrieved from https://json.org/

Kevin Delmolino, A. K., Mitchell Arnett, Miller, A., & Shi, E. (2015). *Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab* (Tech. Rep.). University of Maryland. doi: https://eprint.iacr.org/2015/460.pdf

Krebber, M., Barthels, H., & Bientinesi, P. (2017). *Efficient pattern matching in python* (Tech. Rep.). Aachen Institute for Advanced Study in Computational Engineering Science High-Performance and Automatic Computing Group, RWTH Aachen University. doi: https://arxiv.org/pdf/1710.00077.pdf

Ll parsing. (n.d.). Retrieved from https://en.wikipedia.org/wiki/LL_parser

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. , 254-269. doi: https://dl.acm.org/citation.cfm?id=2978309

Martin, R. C., & Martin, M. (2007). *Agile principles, patterns, and practices in c#.* Pearson Education Inc., 501 Boylston Street, Suite 900, Boston MA 02116: Pearson Education Inc.

Mitchell, R. (2018). *Web scraping with python.* O'Rielly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Rielly Media, Inc.

Nakamoto, S. (2009). *Bitcoin: A peer-to-peer electronic cash system* (Tech. Rep.). doi: www.bitcoin.org

Nicola Atzei, M. B., & Cimoli, T. (2017). *A survey of attacks on ethereum smart contracts* (Tech. Rep.). Università degli Studi Cagliari, Italy. doi: https://eprint.iacr.org/2016/1007.pdf

Pettersson, J., & Edström, R. (2016). *Safer smart contracts through type-driven development.* Gothenburg, Sweden.

Python regex library. (n.d.). Retrieved from https://docs.python.org/3/library/re.html

Python request library. (n.d.). Retrieved from http://docs.python-requests.org/en/master/user/advanced/

Python subprocess library. (n.d.). Retrieved from https://docs.python.org/3/library/subprocess.html

Solidity. (n.d.). Retrieved from https://solidity.readthedocs.io/en/v0.4.25/

Solidity selfdestruct. (n.d.). Retrieved from https://solidity.readthedocs.io/en/v0.4.21/introduction-to-smart-contracts.html

Szabo, N. (1997). *Formalizing and securing relationships on public networks* (Tech. Rep.).

Wikipedia, plagiarism. (n.d.). Retrieved from `https://en.wikipedia.org/wiki/Plagiarism`

Wood, D. G. (2014). *Ethereum: A secure decentralised generalised transaction ledger* (Tech. Rep.). doi: https://gavwood.com/paper.pdf

Ægidius Mogensen, T. (2017). *Introduction to compiler design.* Springer International Publishing AG, Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International Publishing AG.