Google Summer of Code

Creating Pure-Python Fallback Dependencies

Project Proposal GSoC 2023

—

About Project

- Project Size is Large.
- The issue for this Proposal.
- The Idea for this Proposal.

Abstract

- Currently SCTK uses pyahocorasick, intbitset for license detection & lxml for creating cyclonedx-xml output formats.

- All three of them are currently implemented as a Extension of Python using C or Cython.

- These libraries are important for the performance of ScanCode but they need to be compiled and installed in a specific way, which may not be possible on all computers.

- This can make ScanCode less portable or difficult to use for some users who do not have the necessary tools or libraries installed on their machines.

- Therefore to address this issue, the proposal is to create fallback dependencies that are written in Python and can be used by ScanCode when the required C libraries are not available, making it easier to install and use on a wider range of machines.

Project Description

- In all three modules, the plan is to keep the names and parameters of the APIs the same, so that there are very minimal changes in the existing code of Scancode Toolkit.

Pyahocorasick

- This module is currently implemented as a C Extension for Python.

- It implements the ahocorasick algorithm.

- It is used for exact or approximate multi pattern string search.

- With respect to the discussions and feedback received from the mentors, the plan is, instead of implementing the entire thing from scratch, we can build a wrapper around a pre-existing pure python implementation of the ahocorasick algorithm. One of the best options for this would be abusix/ahocorapy.

- One reason ahocorapy library is chosen over any other pure python implementation of ahocorasick such as Guangyi-Z/py-aho-corasick, is because it is highly optimized (benchmarks here) and is pickleable.

Library	Setup(1x)	Search(100x)
ahocorapy	0.30s	0.29
pyahocorasick	0.04s	0.04s
py-aho-corasick	0.72s	4.60s

- Below is sample wrapper around the ahocorapy which allows us to:
    a. Initialize the automaton class.
    b. Add words.
    c. Create an automaton.

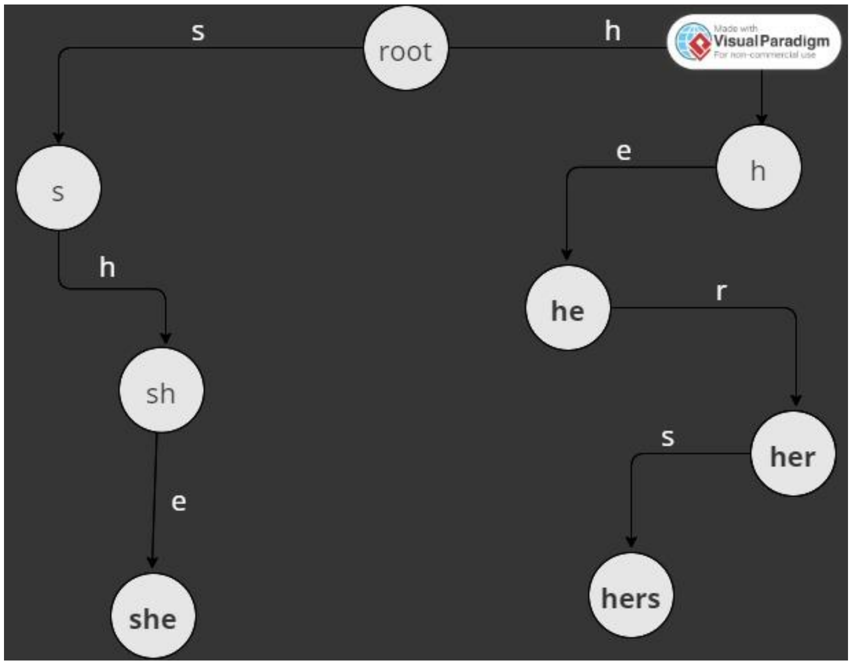
```
from ahocorapy.keywordtree import KeywordTree


class Automaton:
   def __init__(self):
       self.automaton = KeywordTree()
       self.key_value_dictionary = {}


   def add_word(self, key, value=None):
       # abusix/ahocorapy does not give an option to maintain a key value
pair, so we can create a dictionary which stores the key value pair.
       self.automaton.add(key)
       self.key_value_dictionary[key] = value


   def make_automaton(self):
       self.automaton.finalize()
```

- Working of the ahocorasick algorithm:

    - The ahocorasick algorithm creates a finite set machine that resembles a trie with additional links between various internal nodes. These extra internal links allow fast transitions between failed string matches, to other branches of the trie that share a common prefix. This allows the automaton to transition between string matches without the need for backtracking.

    - Trie: Trie data structure is a tree-based data structure used for storing collections of strings. The word trie comes from the word reTRIEval which means to find or get something back. If two strings have a common prefix then they will have the same ancestor in the trie. In a trie data structure, we can store a large number of strings and can do search operations on it in an efficient way.

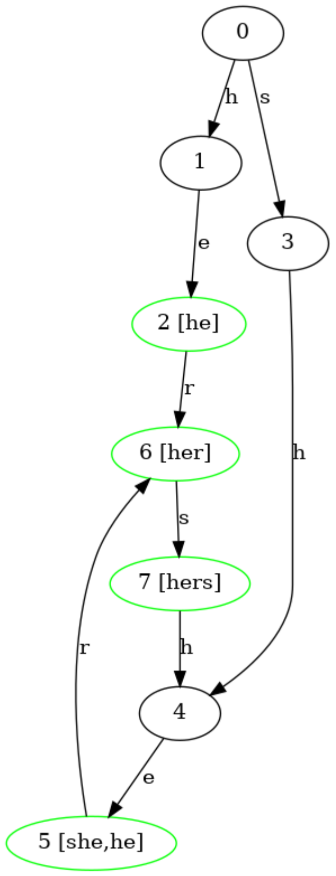
    - Below is a Trie for words ‘he’, ‘she’, ‘her’, ‘hers’.



- Every Node is an instance of class Node:

```python
class Node:
    def __init__(self, letter, value=None):
        self.letter = letter
        self.value = value
        self.children = {}
```

- This **Node Class** above is **analogous** to **ahocorapy's State Class**.
- Automaton for the above Trie.



- The **insert method** below is **analogous** to **ahocorapy's add method**.

```
class Automaton:
    def __init__(self) -> None:
        self.root = Node(None, None)
        self.current_node: Node = self.root
        self.root = Node('')
        self.insert_curr = self.root
        self.search_curr = self.root


    def insert(self, key: str) -> Node:

        self.insert_curr = self.root

        """Key must be strictly an string"""
        key_size = len(key)


        for letter in list(key):

            present = self.insert_curr.children.get(letter, None)
            if present is not None:
                self.insert_curr = present

            #Otherwise
            else:
                new_node = Node(letter=letter)
                self.insert_curr.children[letter] = new_node
                self.insert_curr = new_node

        return self.insert_curr
```

Intbitset

- This module is written in Cython.

- This module uses bit array Data Structure for faster set operations.

- It enables faster set operations such as intersection, union etc.

- Bit arrays are basically array data structures that compactly store bits. It can be used to implement a simple set data structure. A bit array is effective at exploiting bit-level parallelism in hardware to perform operations quickly.

- Example:
    - Set1 = [1, 3, 4, 6]
    - Set2 = [1, 2, 3, 4]
    - If the number is present in the set we represent it as a set bit i.e. 1, otherwise 0.
    - Numbering of bitset starts from rhs to lhs.
    - So, we can use bitset to represent them as:
    - bitSet1 = 1011010
    - bitSet2 = 0011110
    - To find Intersection of Set1 and Set2, we can just do:
    - bitSet1 AND bitSet2 = 0011010
    - Which translates to [1, 3, 4], which is equal to Set1 INTERSECTION Set2.

- Below is a simple implementation of bit array:

```
class BitSet:

   def __init__(self):

       self.bitset = 0


   def add_element(self, integer):

       self.bitset |= 1 << integer
```

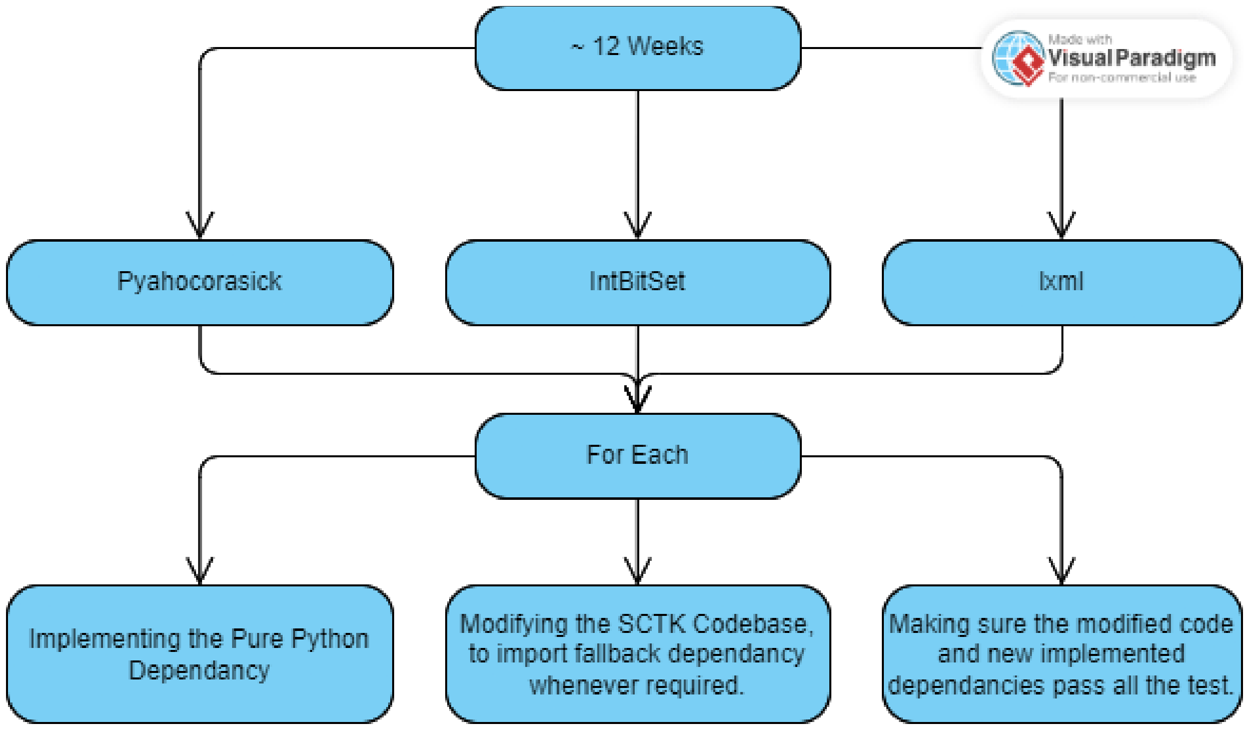
- Below are sample operations to add numbers to bit arrays and perform union and intersection:

```
class intbitset:

   def __init__(self):
       self.bitset = BitSet()

   def append(self, integer):
       if integer < 0:
           raise ValueError("Value Can't be Negative")
       self.bitset.add_element(integer)

   def intersection(self, other):
       self.bitset.bitset &= other.bitset.bitset

   def union(self, other):
       self.bitset.bitset |= other.bitset.bitset
```

lxml

- Lxml is a library that uses libxml2, which makes it really fast.
- libxml2 is an XML toolkit implemented in C.
- This library allows users fast Parsing, validation & operations on XML and HTML.
- In context to SCTK, it is used for creating scan output in XML format (here) and for parsing Maven POMs (here).
- The plan is to build an API using the Python's in-built XML Processing Modules.
- In addition to this we might also have to make changes in sassoftware/pymaven as it uses lxml (here) which in turn is used by SCTK.

Project Timeline


~ 12 Weeks
Made with Visual Paradigm For non-commercial use
Pyahocorasick
IntBitSet
lxml
For Each
Implementing the Pure Python Dependancy
Modifying the SCTK Codebase, to import fallback dependancy whenever required.
Making sure the modified code and new implemented dependancies pass all the test.

Period	Tasks
June 1 - June 10 (10 days)	Will Start to implement pure python dependency for Pyahocorasick, according to plan, during this time frame, we will prepare a wrapper around an existing implementation of aho-corasick such as ahocorapy.
June 11 - June 16 (5 days)	Will make changes in SCTK Codebase to switch to Pyahocorasick fallback dependencies whenever required.
June 17 - June 21 (5 days)	Make sure all the changes and the aho-corasick dependency passes all tests present in SCTK and the pyahocorasick implementation.

June 22 - July 1 (10 days)	Will Start to implement pure python dependency for Intbitset.
July 2 - July 7 (5 days)	Make changes in SCTK Codebase to switch to Intbitset fallback dependencies whenever required.
July 8 - July 13 (5 days)	Make sure all the changes and the intbitset dependency passes all the tests.
July 14	**Mid Term Evaluation**
July 15 - July 25 (10 days)	Start Implementing pure python dependency for lxml. This will be built using the inbuilt Python xml library.
July 26 - August 4 (10 days)	Make changes in SCTK Codebase to switch to lxml fallback dependencies whenever required.
August 5 - August 11 (7 days)	Make sure all the changes and the intbitset dependency passes all the tests.

About Me

Proposal by Jay Kumar

India (IST, GMT+5:30)

Element: @35c4n0r-61d6fbac6da03739848dc19a:gitter.im

GitHub: https://github.com/35C4n0r

I am a second year undergraduate, pursuing my B.Tech in Computer Science Engineering.

I am proficient in Python, C++ & JS.

I will be starting from 1st of July, as my exams will end on 31st of June.

Other than that, I do not have any other Commitments during the GSoC Timeframe.

Contributions

- I have been consistent in attending the Community meetings on Mondays, and

  I have contributions in:

  ### PurlDB:

- https://github.com/nexB/purldb/pull/19 [Merged]

  This PR is related to fixing the email extraction from .dsc files, and adding and fixing related tests.

- https://github.com/nexB/purldb/pull/36 [Merged]

  This PR fixed the documentation that led to an error where make postgres was called without calling make envfile first.

- https://github.com/nexB/purldb/pull/41 [Merged]

  This PR patched the DebianIndexVisitor function, and also added new tests and fixed the related test.

- https://github.com/nexB/purldb/pull/48 [Merged]

  This PR fixed the setup.cfg for the python version specifier.

FetchCode:

- https://github.com/nexB/fetchcode/pull/84

  This PR adds the feature that allows users to get additional rubygems data from purl and package data for different versions too.

- https://github.com/nexB/fetchcode/pull/85

  This PR adds an Optional parameter `all_versions` which is false by default, if passed true returns packageinfo for all versions.

- https://github.com/nexB/fetchcode/pull/89 [Draft PR]

  This PR adds support for Debian Packages in Fetchcode. This is still a Draft and needs additional review and Guidance.

Thank you for considering my proposal for the GSoC program. I want to assure you that I am fully committed to this opportunity. I am not applying to any other organization and will be completely focused on contributing my utmost abilities to your project if selected.