Static Inconsistency Detection of Python Class Inheritance

Problem

- There is a possibility that, given their non-trivial complexities, class hierarchies in commonly-used, open-source Python libraries inadvertently contain inconsistent inheritance, logical or cyclic.
- **Logical inconsistency** is defined by a mismatch of the declared inheritance order of a class's parent classes and the method resolution order (MRO) of one or more of those parent classes.
- If a class that exhibits logical inconsistency is inherited by another class, that class's linearization becomes ambiguous.

```python
class B:
    def __init__(self, x):
        self.x = x

    def print_value(self):
        print(self.x)

class C(B):
    def __init__(self, x):
        self.x = x

    def print_value(self):
        print(self.x * 2)

class A(B, C):
    def __init__(self, x):
        self.x = x

    # ambiguous which print_value is inherited in class A

a = A(5)
a.print_value() # ambiguous as to whether 5 or 10 is outputted here
```

The code snippet contains 3 classes, classes A, B, and C. Class C linearizes as [C, B], but B and C are inherited by class A as B, then C, a mismatch.

- **Cyclic inconsistency** is defined by a cycle of class dependencies in the class hierarchy graph. A cyclic inconsistency indicates cyclic class dependencies.

.../a.py:	.../b.py:	.../c.py:
from b import B	from c import C	from a import A
class A(B): ...	class B(C): ...	class C(A): ...

Assuming that a.py, b.py, and c.py are located in the same path, there is a cyclic inconsistency with classes A, B, and C.

Solution

- A custom static checker that can parse class hierarchy graphs from Python libraries and check for inconsistencies.
- Inspection tool can provide information on the inconsistencies.

Objectives

- Develop the proposed custom static checker.
- Use static checker to check for logical and cyclic inconsistencies in open-source Python libraries and get information on the inconsistencies.
- Get and compare results across a variety of Python libraries.

Approach

1. The Python ast library [4] is used to construct the class hierarchy graph.
   - The class hierarchy graph is implemented as a directed graph:
     - Nodes: class identifiers that consist of the relative module paths, resolved in a manner that mimics Python's import resolution logic [3], appended to the respective class names
     - Edges: for edge (u, v), class identifier u inherits from class identifier v
2. The graph is inspected for inconsistencies.
   - Logical inconsistencies are found by using the graph to attempt to linearize the classes with an implementation of the c3 linearization algorithm [2].

Algorithm 1 Memoized c3 linearization algorithm

```
Require: G(V, E): class hierarchy graph, a directed graph
Ensure: L: hashmap, key is class identifier, value is respective c3 linearization
  L ← empty hashmap
  for c_i ∈ V do
    if c_i ∈ L then
      continue
    end if
    lo ← linearization order computed from DFS of c_i ∈ V s.t. ∀c_i, c_j ∈ lo,
  index(lo, c_i) = index(lo, c_j) - 1 ⟹ c_i ∈ neighbor(c_j)
    for c_j ∈ lo do
      if c_j ∈ L then
        continue
      end if
      b ← neighbor(c_j) in order of bases from corresponding ClassDef
      l_b ← [ ]
      can_compute ← true
      for ∀b_i ∈ b in order do
        if L[b_i] == ⊥ then
          report inherited logical inconsistency
          can_compute ← false
          L[c_j] ← ⊥
          break
        else
          append L[b_i] to l_b
        end if
      end for
      if ¬ can_compute then
        continue
      end if
      append b to l_b
      L[c_j] ← get_c3_linearization(c_j, l_b)
    end for
  end for

function get_c3_linearization(c_i, l_b)
  l_i ← [c_i]
  while |l_b| > 0 do
    c_j ← first head(l_b[k]) s.t. ¬ tail(l_b[k]), k = 0...|l_b| − 1
    if c_j == ⊥ then
      report source logical inconsistency
      return ⊥
    end if
    remove c_j from l_b[k], k = 0...|l_b| − 1
    remove [ ] from l_b
    append c_j to l_i
  end while
  return l_i
end function
```

- If a linearization mismatch is found, i.e. the next class cannot be unambiguously chosen from the linearization order of the parent classes, then a logical inconsistency is found.
- Logical inconsistencies are also found with successive inheritance of logically inconsistent classes. To distinguish, the starting point of a logical inconsistency is considered a "source" logical inconsistency, and following ones that derive from it are considered "inherited" logical inconsistencies.
- Cyclic inconsistencies are found by using Tarjan's algorithm to find strongly connected components [1]. Each corresponding class from the nodes in each multi-node strongly connected component is in a cyclic inconsistency.

Algorithm 2 Tarjan's algorithm to find strongly connected components

```
Require: G(V, E): directed graph
Ensure: sccs: list of strongly connected components, s: stack of nodes possibly
  in a strongly connected component
  sccs ← [ ]
  for v ∈ V do
    indexof(v) ← ⊥
    lowlink(v) ← ⊥
    onStack(v) ← false
  end for
  index ← 0
  s ← empty stack
  for v ∈ V do
    if indexof(v) == ⊥ then
      index ← strongconnect(v, index)
    end if
  end for

function strongconnect(v, index)
  indexof(v) ← index
  lowlink(v) ← index
  index ← index + 1
  add v to s
  onStack(v) ← true
  for w ∈ neighbors(v) do
    if indexof(w) == ⊥ then
      index ← strongconnect(w, index)
      lowlink(v) ← min(lowlink(v), lowlink(w))
    else if onStack(w) then
      lowlink(v) ← min(lowlink(v), indexof(w))
    end if
  end for
  if lowlink(v) == indexof(v) then
    scc ← [ ]
    w ← ⊥
    while w ≠ v do
      w ← pop from s
      onStack(w) ← false
      append w to scc
    end while
    append scc to sccs
  end if
  return index
end function
```

Results

library	scikit-learn	NumPy	PyTorch	Keras	TensorFlow	Matplotlib
number of source logical inconsistencies	0	0	0	0	0	0
number of inherited logical inconsistencies	0	0	0	0	0	0
number of logical inconsistencies	0	0	0	0	0	0
number of cycle inconsistencies	0	0	0	0	0	0
number of resolved bases from ClassDefs	820	469	1870	907	4394	500
number of class definitions	665	1374	3356	969	5381	711

library	SciPy	Theano	PyTensor	pandas	Seaborn	NLTK	Plotly
number of source logical inconsistencies	0	0	0	0	0	0	0
number of inherited logical inconsistencies	0	0	0	0	0	0	0
number of logical inconsistencies	0	0	0	0	0	0	0
number of cycle inconsistencies	0	0	0	0	0	0	0
number of resolved bases from ClassDefs	585	867	429	452	75	542	1122
number of class definitions	1720	1239	621	1539	122	802	10999

Across the thousands of statically resolved class inheritances from various open-source Python libraries, no logical or cyclic inconsistencies have been found likely due to stringent code contribution standards.

Limitations and Future Work

- The custom static inconsistency detector does not track classes that fall under these cases:
  - Classes defined in non-global closure (i.e. within another class or function)
  - Multiple definitions of same named class within the same file
  - Base identifiers that refer to variables set to classes
- Future work on this project can address and attempt to solve these limitations.

Implications

- Baseline MRO verification of class inheritance in Python libraries.
- Static check for circular class inheritance dependencies in Python libraries.
- Proposal of change requests to open-source Python libraries for found logical or cyclic inconsistencies.

References

[1] Robert Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[2] Kim Barrett et al. "A monotonic superclass linearization for Dylan". In: *Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. 1996, pp. 69–82.

[3] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[4] Python Software Foundation. *Abstract Syntax Tree*. URL: https://docs.python.org/3/library/ast.html.