Python 2.3: C3 Method Resolution Order

William Tholke
Foothill College, Intermediate Software Design in Python [CS3B]

May 30, 2021

1 Introduction

As a prerequisite to deciphering the Method Resolution Order in Python 2.3, we need to understand multiple inheritance

Theorem 1 ***Multiple Inheritance*** *is the mechanism through which multiple classes, called subclasses, inherit the methods and properties from any number of parent classes*

Consider the following hierarchy:

```
class Parent1:
    pass

class Parent2:
    pass

class Child(Parent1, Parent2):
    pass
```

In the example above, the *Child* class both inherits and is derived from its parent classes, which begs the question: in what order are the *Base 1* and *Base 2* parent classes inherited from? In more complex inheritance hierarchies, the set of rules which define that order is called the **Method Resolution Order**.

Theorem 2 *The **Method Resolution Order** (MRO) is the set of rules that dictate how child classes inherit methods and properties from a hierarchy of parent classes. The output of the linearization of the youngest child class is the MRO.*

2 Algorithm

The C3 superclass linearization algorithm is the sum of the youngest child class plus both a merge of the linearization of its parents and a list of the parents as the last argument in the merge.

Note that *head* refers to the first element of a list and that *tail* refers to the last element of a list.

Completing the parents' merge is as follows:

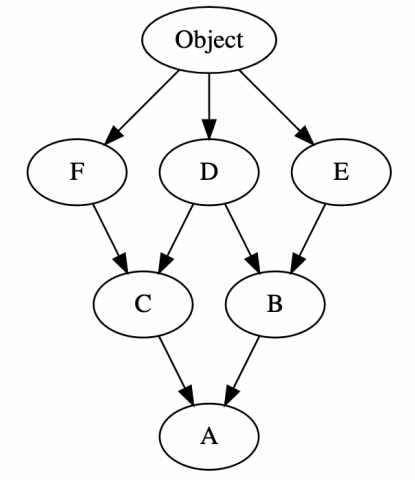
1. Select the first head of the lists that does not appear in the tail of any other list
    - Valid heads may be the first element in multiple lists that do not appear anywhere in any other lists
2. Remove the selected element from all lists where it appears as a head and append it to the output list to be totaled
3. If no valid head can be selected, the merge must halt due to inconsistent dependency ordering in the inheritance hierarchy. In this case, no linearization exists

3 Example

Let's construct the MRO for the following hierarchy:

```
>>> O = object
>>> class F(O): pass
>>> class E(O): pass
>>> class D(O): pass
>>> class C(D,F): pass
>>> class B(E,D): pass
>>> class A(B,C): pass
```

Below is a dependency graph for the example:



The linearizations are calculated as follows:

```
L[O] = O
L[D] = D O
L[E] = E O
L[F] = F O

L[B] = B + merge(EO, DO, ED)
     = B + E + merge(O, DO, D)
     = B + E + D + merge(O, O)
     = B E D O

L[C] = C + merge(DO, FO, DF)
     = C + D + merge(O, FO, F)
     = C + D + F + merge(O, O)
     = C D F O
```

Finally, we can calculate the linearization of the youngest child class A:

```
L[A] = A + merge(BEDO, CDFO, BC)
     = A + B + merge(EDO, CDFO, C)
     = A + B + E + merge(DO, CDFO, C)
     = A + B + E + C + merge(DO, DFO)
```

```
= A + B + E + C + D + merge(O, FO)
= A + B + E + C + D + F + merge(O, O)
= A B E C D F O
```

In the above example, the MRO is ordered as **A B E C D F O**.