C3 linearization

From Wikipedia, the free encyclopedia

In computing, the **C3 superclass linearization** is an algorithm used primarily to obtain the order in which methods should be inherited (the "linearization") in the presence of multiple inheritance, and is often termed "**MRO**" for **Method Resolution Order**. The name C3 refers to the three important properties of the resulting linearization: a consistent extended precedence graph, preservation of local precedence order, and fitting the monotonicity criterion. (The name "C3" is not an initialism.) It was first published at the 1996 OOPSLA conference, in a paper entitled "A Monotonic Superclass Linearization for Dylan".[1] It was adapted to the Open Dylan implementation in January 2012[2] following an enhancement proposal.[3] It has been chosen as the default algorithm for method resolution in Python 2.3 (and newer),[4][5] Perl 6,[6] and Parrot.[7] It is also available as an alternative, non-default MRO in the core of Perl 5 starting with version 5.10.0.[8] An extension implementation for earlier versions of Perl 5 named `Class::C3` exists on CPAN.[9]

Description [edit]

The C3 superclass linearization of a class is the sum of the class plus a unique merge of the linearizations of its parents and a list of the parents itself. The list of parents as the last argument to the merge process preserves the local precedence order of direct parent classes.

The merge of parents' linearizations and parents list is done by selecting the first head of the lists which does not appear in the tail of any of the lists. Note, that a good head may appear as the first element in multiple lists at the same time, but it is forbidden to appear anywhere else. The selected element is removed from all the lists where it appears as a head and appended to the output list. The process of selecting and removing a good head to extend the output list is repeated until all remaining lists are exhausted. If at some point no good head can be selected, because the heads of all remaining lists appear in any one tail of the lists, then the merge is impossible to compute due to cyclic dependencies in the inheritance hierarchy and no linearization of the original class exists.

A naive divide and conquer approach to computing the linearization of a class may invoke the algorithm recursively to find the linearizations of parent classes for the merge-subroutine. However, this will result in an infinitely looping recursion in the presence of a cyclic class hierarchy. To detect such a cycle and to break the infinite recursion (and to reuse the results of previous computations as an optimization), the recursive invocation should be shielded against re-entrance of a previous argument by means of a cache or memoization.

Example [edit]

Given

```
class O
class A extends O
class B extends O
class C extends O
class D extends O
class E extends O
class K1 extends A, B, C
class K2 extends D, B, E
class K3 extends D, A
class Z extends K1, K2, K3
```

the linearization of Z is computed as

```
L(O)  := [O]                                                    // the
linearization of O is simple the singleton list [O], because it has no parents

L(A)  := [A] + merge(L(O), [O])                                 // the
linearization of A is A plus the merge of its parent's linearization and the list of
parents...
       = [A] + merge([O], [O])
```

Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Русский
Edit links

```
       = [A, O]                                                    // ...which
trivially prepends A to its parent's linearization

L(B)  := [B, O]                                                    // linearizations
of B, C, D and E are computed similar to A
L(C)  := [C, O]
L(D)  := [D, O]
L(E)  := [E, O]

L(K1) := [K1] + merge(L(A), L(B), L(C), [A, B, C])                 // first, find the
linearizations of A, B and C and merge them with the list [A, B, C]
       = [K1] + merge([A, O], [B, O], [C, O], [A, B, C])           // class A is a
good candidate in the merge, because it only appears as the head of the first and
last lists
       = [K1, A] + merge([O], [B, O], [C, O], [B, C])          // class O is no good
candidate in the merge, because it also appears in the tails of list 2 and 3,
       = [K1, A, B] + merge([O], [O], [C, O], [C])             // found B as a good
head in the merge, so it is removed from the lists and appended to the output
       = [K1, A, B, C] + merge([O], [O], [O])
       = [K1, A, B, C, O]

L(K2) := [K2] + merge(L(D), L(B), L(E), [D, B, E])
       = [K2] + merge([D, O], [B, O], [E, O], [D, B, E])
       = [K2, D] + merge([O], [B, O], [E, O], [B, E])
       = [K2, D, B] + merge([O], [O], [E, O], [E])
       = [K2, D, B, E] + merge([O], [O], [O])
       = [K2, D, B, E, O]

L(K3) := [K3] + merge(L(D), L(A), [D, A])
       = [K3] + merge([D, O], [A, O], [D, A])
       = [K3, D] + merge([O], [A, O], [A])
       = [K3, D, A] + merge([O], [O])
       = [K3, D, A, O]

L(Z)  := [Z] + merge(L(K1), L(K2), L(K3), [K1, K2, K3])
       = [Z] + merge([K1, A, B, C, O], [K2, D, B, E, O], [K3, D, A, O], [K1, K2,
K3])    // select K1
       = [Z, K1] + merge([A, B, C, O], [K2, D, B, E, O], [K3, D, A, O], [K2, K3])
// fail A, select K2
       = [Z, K1, K2] + merge([A, B, C, O], [D, B, E, O], [K3, D, A, O], [K3])
// fail A, fail D, select K3
       = [Z, K1, K2, K3] + merge([A, B, C, O], [D, B, E, O], [D, A, O])
// fail A, select D
       = [Z, K1, K2, K3, D] + merge([A, B, C, O], [B, E, O], [A, O])
// select A
       = [Z, K1, K2, K3, D, A] + merge([B, C, O], [B, E, O], [O])
// select B
       = [Z, K1, K2, K3, D, A, B] + merge([C, O], [E, O], [O])
// select C
       = [Z, K1, K2, K3, D, A, B, C] + merge([O], [E, O], [O])
// fail O, select E
       = [Z, K1, K2, K3, D, A, B, C, E] + merge([O], [O], [O])
// select O
       = [Z, K1, K2, K3, D, A, B, C, E, O]
// done
```

References [edit]

1. ^ "A Monotonic Superclass Linearization for Dylan". *OOPSLA '96 Conference Proceedings*. ACM Press. 1996-06-28. pp. 69–82. doi:10.1145/236337.236343. ISBN 0-89791-788-X
2. ^ News item on opendylan.org
3. ^ Dylan Enhancement Proposal 3: C3 superclass linearization
4. ^ Python 2.3's use of C3 MRO
5. ^ Tutorial for practical applications of C3 linearization using Python
6. ^ Perl 6's use of the C3 MRO
7. ^ Parrot uses C3 MRO
8. ^ C3 MRO available in Perl 5.10 and newer
9. ^ Perl 5 extension for C3 MRO on CPAN

This programming-language-related article is a stub. You can help Wikipedia by expanding it.

Categories: Object-oriented programming | Programming language implementation | Programming language topic stubs