УДК 004.655

Буй Д.Б.[1] д.-р. фіз.-мат. н., професор,
Компан С.В.[2], Поляков С.А.[3] канд. фіз.-мат. н.,
Карам Д.[4] аспірант

Алгоритми CLOS та LOOPS лінеаризації класів в мовах програмування: формальна побудова

Київський національний університет імені Тараса Шевченка, 03680, м. Київ, пр-т. Глушкова 4д

e-mail: [1] buy@unicyb.kiev.ua
e-mail: [2] robin_2005@mail.ru
e-mail: [3] sergey.a.polyakov@gmail.com
e-mail: [4] karamjasim1978@yahoo.com

D.B. Buy[1] professor, S.V. Kompan[2],
S.A. Polyakov[3] PhD, J. Karam[4] postgraduate

Linearization algorithms CLOS and LOOPS of the classes in programming languages: the formal definitions

[1]Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova st., 4d

e-mail: [1] buy@unicyb.kiev.ua
e-mail: [2] robin_2005@mail.ru
e-mail: [3] sergey.a.polyakov@gmail.com
e-mail: [4] karamjasim1978@yahoo.com

В статті досліджуються методи вирішення конфліктів імен в мовах програмування CLOS та LOOPS, які дозволяють використовувати множинне успадкування. Ці методи засновані на лінеаризації класів, яка полягає в трансформації ієрархії класів з множинним успадкуванням в ієрархію одиночного успадкування, при якому проблема вибору відповідного атрибута вирішується тривіально. Вводиться визначення ієрархії класів.

Ключові слова: множинне успадкування, CLOS, LOOPS, алгоритми лінеаризації

The methods of conflict resolution of names in programming languages CLOS and LOOPS that allow multiple inheritance were researched. Those methods are based on the linearization of the classes. The idea of linearization is to reduce multiple inheritance to single, in which the problem of selecting a suitable attribute is solved trivially. The definition of a hierarchy of classes is introduced. The property of monotonicity was clarified. The idea of monotonicity is that all the attributes that the class inherits either identified in its direct ancestors, or inherited them. Methods of conflict resolution used in programming languages CLOS and LOOPS were formally defined.

Key Words: Multiple objects inheritance, CLOS, LOOPS, linearization algorithms.

Статтю представив д.ф.-м.н., професор Анісімов А.В.

Introduction. In object-oriented programming languages that implement multiple inheritance, the situation when the same method or field is found in multiple parent classes can take place. In this case, there is the problem of choosing the "right" method or field from several possible. Since the respective algorithms perform equally for both fields and methods of classes, we will in further call fields and methods of class as attributes. If the class $\boldsymbol{X}$ contains an attribute $\boldsymbol{atr}$, then we say that the attribute $\boldsymbol{atr}$ is defined in class $\boldsymbol{X}$ ($\boldsymbol{atr} \in \boldsymbol{X}$).

There are two basic approaches in the resolution of the conflict of names [1]. In the first case, the conflict is resolved trivially by explicit the indication of the class in which the desired attribute exists. This is, for example, in C ++. In the second case, a special algorithm is used to select the "most appropriate" attribute. This approach is used in languages such as CLOS, LOOPS, Python, Perl, Dylan and others. The basic idea is that the parent classes are linearly ordered as the list. Then the method takes the first class in the list in which the attribute is defined.

We can say that the multiple inheritance by means of such ordering, boils down to the single inheritance, in which the problem of name conflict is solved by the fact that each class defines its own scope of visibility, which overload the scope of visibility of its the parent classes.

The main advantage of the second approach is that the name conflict can be resolved dynamically, i.e. during the program execution. Considered technique is called classes' linearization. There are several algorithms to resolve a name conflict by means of linearization. We will consider algorithms used in programming languages CLOS and LOOPS.

Although the algorithms have been created in the 90-th years of the last century, their formal properties are not well understood. That does not allow to talk about creating safe programs in programming languages that use these algorithms for classes' linearization. In addition, when formal definition of the semantics of programs is given, different approaches are usually consider the semantics of basic programming constructions like branching, looping, sequencing, but the semantics of classes and objects are omitted. The obtained results form the basis for a formal model that describes the classes, inheritance and relationships between classes.

The purpose of this paper is to give the formal definitions of these linearization algorithms that make possible the second step – proofing the formal correctness of these algorithms. Other algorithms for linearization examined in [4, 5, 6].

Definitions. Let's introduce the needed definitions [1], [2]. Class X, which is directly inherited from class Y, will be called direct descendant of the class Y, and Y, respectively, the direct ancestor (parent) of X (see. Fig. 1).

Fig. 1. Direct inheritance: Class Y is the direct ancestor (parent) of the class X; the class X is the direct descendant of the class Y

Such inheritance will be written in the form of a pair (X, Y), i.e. the pair (a descendant, an ancestor). A class can have a few direct ancestors, for example, a pair $(X, (Y, Z))$ indicates that the class X inherits from classes Y and Z, wherein the set of ancestors is ordered. Ordered family of items will be called tuples. Graphically, described multiple inheritance is shown in Fig. 2.

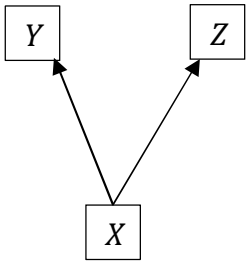

Fig. 2. Multiple inheritance: class X is a direct descendant of classes Y, Z, and direct ancestors are ordered from left to right

In turn classes Y and Z may be inherited from other classes, etc. Thus, we obtain a class hierarchy. It is said that two pairs of direct inheritances $(X_0, (X_1, \dots, X_n))$ and $(Y_0, (Y_1, \dots, Y_m))$ are linked if there exists $i = 1..n$, that $X_i = Y_0$. If $X_i = X_0$, the inheritance is called the self-linked. Graphically, the linked pairs of direct inheritances are shown in Fig. 3, self-linked inheritance is shown in Fig. 4.

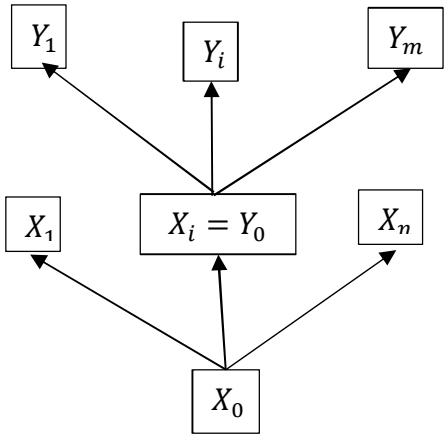

Fig. 3. Linked pairs of direct inheritances

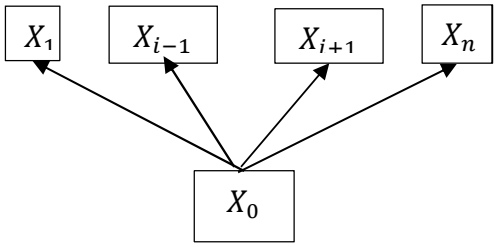

Fig. 4. Self-linked direct inheritance

Sequence of direct inheritances $r_1, r_2, \dots, r_k$ called a cycle when each i-th element is linked with the (i+1)-th for $i = 1..k-1$ and the k-th element is linked with the first.

Let us denote by π_1^2 projection by the first component of the set of ordered pairs, and by π_2^2 – projection by the second component.

Let us denote by Δ the set of the classes, and by $\Delta_{<>}$ the set of all classes of tuples Δ. Class hierarchy is called a binary relation $H \subseteq (\Delta \times \Delta_{<>})$, such that:

- H is functionally, i.e. for each $X \in \pi_1^2(H)$ exists exactly one tuple $(X_1, \dots, X_n) \epsilon \pi_2^2(H)$ that sets its direct ancestors;
- all the elements of the H are not self-linked;
- elements of the H does not form a cycles;
- elements from all tuples of projection $\pi_2^2(H)$ are not repeated.

The class hierarchy defines an acyclic direct simple graph whose vertices are the classes and the edges are defined as follows. For each direct inheritance $(X_0, (X_1, \dots, X_n))$ of the class hierarchy

the set of edges from X_0 will be defined as the set of pairs $\{(X_0, X_1), (X_0, X_2), \dots, (X_0, X_n)\}$. Union of the all of these sets will be the set of edges of the graph. Such graphs will be called the direct inheritance graphs. Each class hierarchy corresponds to a single graph of direct inheritance. But each direct inheritance graph corresponds, in general, several hierarchies of classes. They differ by the order of the elements in a tuple, since during the transition from a pair $(X_0, (X_1, \dots, X_n))$ to the set of pairs $\{(X_0, X_1), (X_0, X_2), \dots, (X_0, X_n)\}$ information about ordering of the classes $X_1, X_2, \dots, X_n$ lost.

The set of all edges of direct inheritance defines the anti-reflective and antisymmetric binary relation on classes. Its transitive closure is a strict partial order relation, denoted by $<_H$.

The relation $X <_H Y$ means that in the graph of direct inheritance there is a path from the class X to the class Y. In this case we say that the class Y is an ancestor of class X, and class X, respectively, a descendant of the class Y.

A linearization can be considered as a topological sorting (see. eg., [3], section 2.2.3) on a graph of direct inheritance or, another words, it specify linear order $<_\tau$ on the set of classes that would include a partial order $<_H$.

Such a topological sort is called linear extension of the graph of direct inheritance. Then we call linearization a unary function L, displaying a graph of direct inheritance of H in its linear extension $L(H)$. Thus, topological sorting is an example of a linearization.

The idea of linearization is, in fact, the reduction a multiple inheritance to a single inheritance, in which the problem of choosing the most suitable attribute from several ones is solved trivially.

For the class Y denote via H_Y the subgraph of direct inheritance graph of the hierarchy H, consisting of class Y and all of its ancestors (not only direct).

Under the method of conflict resolution we mean the mapping M, assigning to the class X, direct inheritance graph of the hierarchy H, and attribute atr the class Y from subgraph H_X, such that attribute atr is defined in this class: $X, H, atr \xrightarrow{M} Y$, attribute atr is defined in class Y. In particular, if the attribute atr is defined in the class X, then $M(X, H, atr) = X$.

The method of conflict resolution is not bound only to linearization of the direct inheritance graph and any other mapping satisfying definition can be used.

Adequate methods of conflict resolution must satisfy a number of natural properties, which essentially is not to produce results contrary to intuition.

The main feature is the so-called monotonicity. The idea is that all the attributes that a class inherits are defined in its direct ancestors, or inherited them. I.e. situation is impossible when a class inherits an attribute, but none of his direct ancestors; this attribute does not inherit or not defined. Monotonicity means that the class can be considered as a specialization (specification) of his direct ancestors, i.e. it has all the attributes that his direct ancestors have, plus the additional attributes that are unique to this class.

Methods of conflict resolution M is monotone if for any inheritance graph $H = (C, I)$, any class $X \in C$ and any attribute $atr \in X$ either $M(X, H, atr) = X$ or $M(X, H, atr) = Y, Y \neq X$, and there exists at least one direct ancestor Z of class X, such that $atr \in Z \,\&\, M(Z, H, atr) = Y$. In particular, it can be a situation that $Y = Z$.

We proceed to the definition of the monotonicity feature for linearization. Linearization L is monotone if for any inheritance graph $H = (C, I)$, and for any classes $X, Y \in C$ such that X is an ancestor of the Y, $L(H_X)$ included in the $L(H_Y)$.

Let's introduce the relation $pred_C$ of the precedence of classes on a hierarchy of classes H, where C is a set of classes. Two classes X_1 and X_2 are in relation of precedence if there is a direct inheritance $(Y_0, (Y_1, \dots, Y_n)) \in H$, that $X_1 = Y_i$ and $X_2 = Y_j$ and $i < j$, $i, j = 1..n$.

Relation of the precedence can be obtained as the union of all the relations of local linear orders, obtained by ordering direct ancestor classes.

The relation $pred_C$ must be acyclic for correct algorithms work. Then for each class you can specify an ordered set of his previous classes.

Linearization algorithms. Let's go directly to the linearization algorithm.

Algorithm of the linearization LOOPS.

For the hierarchy of the classes H build a direct inheritance graph $H = (C, I)$ and define relation of the precedence $pred_C$. Algorithm is defined in the triple $(C, I, pred_C)$, called the graph representation of the direct inheritance and denoted $R(H)$. It builds a linearization of the direct inheritance graph using the precedence relation to resolve the ambiguity. We denote this the linearization as $L(R(H))$.

For class X and representation graph of the inheritance $R(H_X)$ algorithm works as follows.

1) As an initial linearization it takes a sequence consisting of single element $L_X = [X]$.

2) We examine L_X from right to left and look for the first class X, which has at least one direct ancestor such that all its descendants already have been chosen, and it is not already chosen. If X has

few such direct ancestors, then take the first with respect to the relation $pred_C$. Add it to the sequence L_X.

3) Repeat step 2 until it is possible.

Algorithm CLOS.

1) As an initial linearization it takes a sequence consisting of a single element $L_X = [X]$.

2) We examine L_X from right to left and look for the first class, in which there is a direct ancestor such that all its descendants and all previous with respect to relation $pred_C$ classes have already been chosen, and it is not already selected. Add it to the sequence L_X.

3) Repeat step 2 until it is possible.

Conclusion. The paper clarified multiple inheritance classes (object-oriented): direct inheritance, linked couples, and cycles of direct inheritance, a class hierarchy, the graph of direct inheritance hierarchy, its transitive closure and linearization method of conflict resolution and its monotonicity. In these formal terms we specified linearization algorithms LOOPS and CLOS. The formal specification allows the formal proof of the correctness of these algorithms. It is the purpose of the subsequent work.

Список використаних джерел

1. *Ducournau R.* Proposal for a Monotonic Multiple Inheritance Linearization / R. Ducournau, M. Habib, M. Huchard and M. L. Mugnier // OOPSLA'94 Proceedings of the ninth annual conference on Object-oriented programming systems, languages, and applications, 1994, P. 164–175.

2. *Ducournau R.* Monotonic conflict resolution mechanisms for inheritance / R. Ducournau, M. Habib, M. Huchard and M. L. Mugnier // OOPSLA'92 Proceeding conference proceedings on Object-oriented programming systems, languages, and applications, 1992, P. 16–24.

3. *Knuth D.E.* The art of computer programming. Vol. 1. Fundamental Algorithms / D.E. Knuth. – Addison-Wesly Publishing, Inc., 1998.

4. *Simionato M.* The Python 2.3 Method Resolution Order [Електронний ресурс] / M. Simionato – Режим доступу до ресурсу: https://www.python.org/download/releases/2.3/mro/

5. *Guido V. R.* Method Resolution Order [Електронний ресурс] / van Rossum Guido. – 2010. – Режим доступу до ресурсу: http://python-history.blogspot.com/2010/06/method-resolution-order.html.

6. *Par Gaël Pegliasco* Python Tutorial: Understanding Python MRO – Class search path [Електронний ресурс] / Pegliasco Par Gaël. – 2014. – http://makina-corpus.com/blog/metier/2014/python-tutorial-understanding-python-mro-class-search-path

References

1. Ducournau, R., Habib, M., Huchard, M. and Mugnier, M.L. (1994) Proposal for a Monotonic Multiple Inheritance Linearization. *Proceedings of the ninth annual conference on Object-oriented programming systems, languages and applications*, OOPSLA'94, pp. 164–175, 1994.

2. Ducournau, R., Habib, M., Huchard, M. and Mugnier, M.L. (1992) Monotonic conflict resolution mechanisms for inheritance. *Proceeding conference proceedings on Object-oriented programming systems, languages, and applications*, OOPSLA'92, pp. 16–24. 1992.

3. Knuth, D.E. (1998) The art of computer programming. Vol. 1 Fundamental Algorithms Addison-Wesly Publishing, Inc., 1998.

4. Michele, Simionato The Python 2.3 Method Resolution Order" [Електронний ресурс]. – Режим доступу до ресурсу: https://www.python.org/download/releases/2.3/mro/

5. Guido, van Rossum (2010) Method Resolution Order [Електронний ресурс]. – Режим доступу до ресурсу: http://python-history.blogspot.com/2010/06/method-resolution-order.html June

6. Par Gaël, Pegliasco (2014) Python Tutorial: Understanding Python MRO – Class search path [Електронний ресурс]. – Режим доступу до ресурсу: http://makina-corpus.com/blog/metier/2014/python-tutorial-understanding-python-mro-class-search-path

Надійшла до редколегії 16.03.15