ON SOME ALGORITHMS FOR MULTIPLE INHERITANCE IN OBJECT ORIENTED PROGRAMMING

R. DUCOURNAU[*], *M. HABIB*[**]

[*]SEMA-METRA
16-18 rue Barbès,
92126 Montrouge cédex.

[**]LIB (Laboratoire commun ENST Br et UBO)
Département Informatique et Réseaux,
Ecole Nationale Supérieure des Télécommunications de Bretagne,
ZI de Kernevent, BP 832,
29285 Brest cédex.

ABSTRACT

We study here some problems yielded by multiple inheritance in object-oriented languages. We give an interpretation and a formalism of heritance mechanisms within the framework of partially ordered sets theory. An inheritance mechanism can be regarded as a traversing algorithm of the inheritance graph, and we prove that those which yields linear extension (total order) play a central role.

We discuss some operational semantic aspects of multiple inheritance, with the introduction of a concept named by *multiplicity.* After a presentation of some well-known inheritance algorithms we propose two new algorithms based upon depth-first search techniques and some particular classes of linear extensions, recently studied.

We end by applying these results in the case of inheritance with exceptions and by setting a few problems. All these results are implemented in the frame-based language YAFOOL[1].

Keywords : *Object oriented language, multiple inheritance, partial ordered sets, linear extensions, depth-first-search, inheritance with exceptions.*

1 *Yet Another Frame-based Object Oriented Language.* YAFOOL was developped by SEMA-METRA, with the participation of INRIA, under contract with the "Centre de Programmation de la Marine", and is the basis of Y3®, the expert systems toolbox of SEMA-METRA.

I INTRODUCTION

Nowadays the paradigms of object-oriented programming are very fruitful and popular in software engineering for prototyping (see GOLDBERG and ROBSON [1983] or BEZIVIN [1986]) or even in classical languages (namely ADA or EIFFEL (see MEYER [1986])) and also in artificial intelligence for representing real-world knowledge (see for example the notion of semantic networks in FAHLMAN [1979]). Among these paradigms, inheritance is a way of sharing knowledge between objects : namely values, attributes, methods and reflexs or deamons can be inherited.

We can distinguish three main aspects in an inheritance mechanism :

1. An interpretation.
2. An operational semantic. (How it works).
3. An inheritance algorithm. (How the inheritance is computed).

There exist many interpretations of inheritance, see BRACHMAN [1983] for a survey, and LIEBERMAN [1986] for a discussion between inheritance and delegation, and BRIOT [1985] for a discussion on the relationships between instanciation and inheritance. We were mainly concerned with the second and third aspects of inheritance. When inheritance is allowed it is quite natural to enlarge it to multiple inheritance. Although with simple inheritance (where the hierarchy of an object is simply a path and the inheritance graph only a tree), the inheritance algorithms are very simple and well defined, it is not so clear when multiple inheritance is allowed.

In this paper we shortly present our results about multiple inheritance algorithms, in the framework of partially ordered sets theory.

I.1 SOME USEFUL GRAPHS AND ORDERED SETS NOTATIONS.

Let X be a finite set and R a binary relation on X, we shall denote by
$G = (X,R)$ the graph of this relation. Elements of X are called *vertices* and those of R, denoted by (x,y) are called *arcs*. Furthermore x and y are respectively the origin and extremity of this arc, y is also called a *successor* of x and $R(x)$ denotes the set of all successors of x in G.
A *path* $[x_1,...,x_k]$, of length $k>1$, is an ordered sequence of vertices such that
$\forall\ i\in [1,k-1],\ (x_i,x_{i+1}) \in R$. This path is a *cycle* if $x_k = x_1$.
An $(x,y) \in R$ is a *transitivity* arc if there exists at least one path of length strictly greater than one going from x to y in G. Such an arc is also called *redundant.*

When G is *acyclic* (without cycle) then its transitive and reflexive closure yields a partial order *(poset* for short) denoted by $\leq_R$.
For $Y \subseteq X$, R/Y denotes the restriction of R to Y, and $G/Y = (Y,R/Y)$ the induced subgraph. Finally, R^d the *dual* of R, satisfies $(x,y) \in R$ iff $(y,x) \in R^d$.

II MULTIPLE INHERITANCE SYSTEMS AND POSETS.

From all the interpretations of inheritance systems (logical systems, set theory, ...) we can assume that an inheritance systems yields a partially ordered relation on the set of objects. More precisely, we can define :

DEFINITION 1 : An inheritance graph is a directed acyclic graph $G = (X,H,\omega)$ where X is the universe of objects, H the inheritance relation which transitive closure is a partially ordered set and ω which is the unique anti-root (or root in the inheritance terminology) of H.

In object oriented systems terminology, ω is called the universe root, and if $(x,y) \in H$ (resp. $x \leq_H y$), y is called a *father* (resp. an *ancestor),* and x, in both cases, is called a *descendant* of y.

In our applications we are only concerned by the following central problem : *For a given object, determine its inheritance.*

DEFINITION 2 : For an object $a \in X$, its inheritance graph (also called its *hierarchy)* is $G(a) = (X_a, H/X_a)$, where $X_a = \{x \in X, a \leq_H x\}$, the set of its ancestors (including a itself). In partial order terminology X_a is the upper ideal of a.

Now we can state our first inheritance principle :

PRINCIPLE 1 : An inheritance mechanism must follow the inheritance partial order.

In particular, the root ω is always the last vertex to be considered.

DEFINITION 3 : An inheritance search is a total order φ_a on X_a.

An object a inherits of a property P, with respect to an inheritance search φ_a, in the following way :

1. P(a) is the first value encountered following φ_a, in a vertex of X_a that admits this property, if there exists any.

2. If there is no such vertex, then a has not the property P. *(This is our closed world assumption).*

This definition yields *hidding effects,* i.e. if x and y admits a property P, and $x \leq_H y$, then P(y) is hidden by P(x). This can be also understood in terms of default, each property P of an object is a default value for all its descendants in the inheritance graph, which admits this property.

DEFINITION 4 : The inheritance mechanism is a mapping : $\varphi : a \in X \to \varphi_a$.

In the following, in order to build such mapping, we have to precise the operational semantic of inheritance we play with.

PROPERTY 1 : The inheritance searches must be stable under redondancy.

We notice that the above property is not essential, we could as well have chosen the opposite and so used the redundancy arcs as a programming control tool.

PRINCIPLE 2 : (uniformity) The inheritance is an uniform mechanism, and its searches apply identically for all object properties.

II.1 MULTIPLICITY.

If an object $a \in X$ has two fathers b *and then* c in the inheritance graph, it is natural to interprete this order as a priority : *a inherits more from b than from c.*

DEFINITION 6 : We define by $\mu(a)$ *the multiplicity relative to a* the total order relation in $H(a)$, the set of fathers of a, and the *multiplicity*, the mapping : $\mu : a \in X \rightarrow \mu(a)$.

This notion is implicit in every graph or poset representation in machine, and moreover is used in every graph traversing algorithm, so we propose to use it as another operational programming control tool (in some way there is an analogy with the implicit ordering of the litterals of a PROLOG clause). Thus an inheritance graph is $G=(X,H,\omega,\mu)$, and moreover we denote by M_a the union of all transitive closures of the $\mu(x)$ for x in X_a and by $M = (X_a,M_a)$ the resulting graph.

PRINCIPLE 3 : (inheritance versus multiplicity) In any case the inheritance relation excels the multiplicity.

For an object a, we say that the multiplicity contradicts the inheritance when the graph $G(a) \cup M=(X_a,H/X_a \cup M_a)$ has a cycle. This may happen in many cases as can be seen in figure 1.

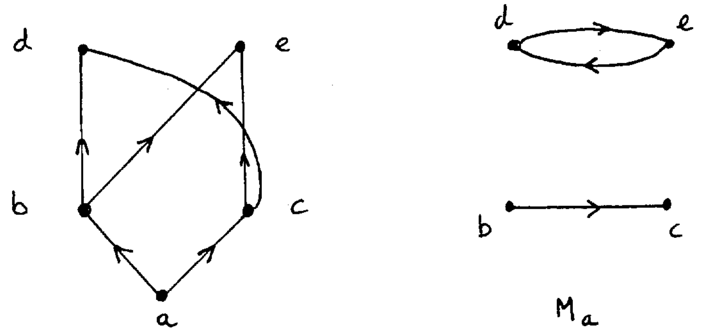


Fig. 1 : Contradictions between multiplicity and inheritance.

Remark : In all the figures, the decreasing multiplicity is drawn from left to right.

PRINCIPLE 4 : When there is no contradiction between multiplicity and inheritance the inheritance search must follow the partial order $H \cup M_a$ yielded by the graph $G(a) \cup M=(X_a,H/X_a \cup M_a)$.

The above principle only generalizes the first principle to multiplicity. But it is still not enough to completely determine the operational semantic of inheritance, we may add a further constraint such as :

PROPERTY 2 : For every object $a \in X$, if b and c are two fathers of a, with $(b,c) \in M_a$, then a must inherit from X_b-X_c before X_c-X_b, with no condition on $X_b \cap X_c$.

As well as property 1, this second property is also arbitrary and can be understood as a depth-first-search choice. We shall examine in detail this property in the following section.

III LINEAR ALGORITHMS.

III.1 SOME KNOWN INHERITANCE SEARCHES.

Most object oriented languages, which allow multiple inheritance, such as for example Common Loops (BOBROW et al. [1986]), or Flavors (MOON [1986]), or Le_Lisp, (CHAILLOUX et al. [1986]), use a depth-first search of the inheritance graph as inheritance search. These traversing graph algorithms are well known since the work of TARJAN [1972]. Some others use a breadth-first search technique like MERING (FERBER [1983]) and even some use some composition of these two searches.

III.1.1 Depth-first search according to multiplicity.

Such a technique always violates our principle 1, since the root ω is not considered last, as can be seen in figure 2.

III.1.2 Breadth-first search according to multiplicity.

Similarly in the right graph of figure 2, by adding a new vertex d, we obtain a counter-example for this search.

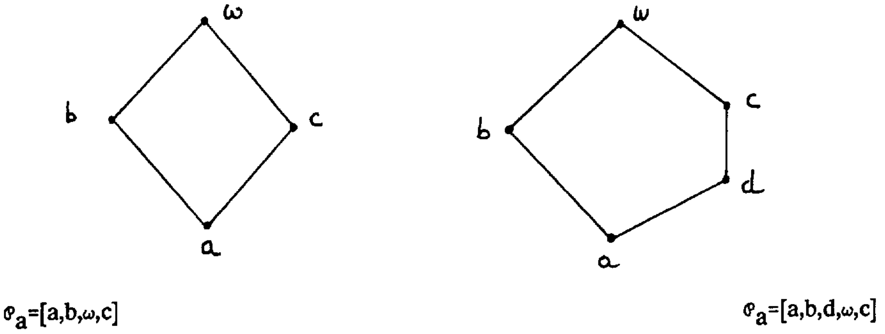


Fig. 2 : contradictions of "naive" searches.

III.2 LINEAR EXTENSIONS.

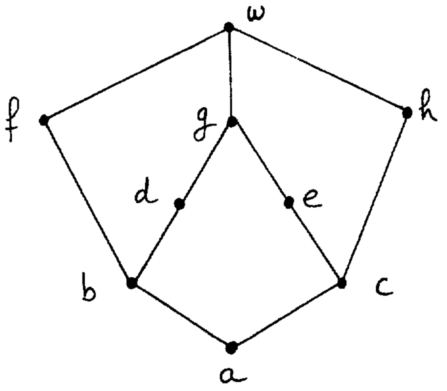
Following TOURETZKY [1986] and AIT-KACI and NASR [1985] we think that it is worth to dig inside partially ordered theory in order to analyze inheritance problems. It is not the only possible approach that can be done, see DUGERDIL [1986] for a completely different point of view.

Let us now, to this aim, introduce a few algorithmic concepts in posets.

DEFINITION 8 : Let $P=(\leq_P,X)$ be a poset :

1. A total order τ on X, is a *linear extension* of P, if $x \leq_P y \Rightarrow x \leq_\tau y$.
2. A linear extension τ of P is *greedy* (resp. *depth-first greedy (dfgreedy for short))* if it can be obtained by application of the following rules :
   a) Choose for x_1 any minimal element of P.
   b) If $x_1,\dots,x_i$ is greedy (resp. dfgreedy), then choose for x_{i+1} any minimal element of $P_i = P - \{x_1,\dots,x_i\}$ covering x_i, (resp. covering x_k where $k \leq i$ is the greatest subscript possible), if there exists one, otherwise choose any minimal element of P_i.

In other words the rule *"take a minimal element and climb as high as you can"* yields the greedy linear extensions. These linear extensions were introduced by COGIS and HABIB [1979], and have now been very well studied (mostly for scheduling problems). Dfgreedy linear extensions were recently introduced by PRETZEL [1985], and BOUCHITTE et al. [1985] have studied some of their algorithmic properties. Figure 3 gives some examples of such linear extensions.



[a,b,c,h,e,g,ω,f,d] is not a linear extension.
[a,b,c,d,e,f,g,h,ω] is a linear extension but not greedy.
[a,b,f,c,h,e,d,g,ω] is a greedy linear extension but not dfgreedy.
[a,b,f,d,,c,h,e,g,ω] is a dfgreedy linear extension.

fig. 3 : greedy and dfgreedy linear extensions.

Let us now formulate our inheritance problem in this framework :

Principle 1 : φ_a is a linear extension of H/X_a.

Principle 4 : φ_a is a linear extension of $H/X_a \cup M_a$.

Property 2 : φ_a is a dfgreedy linear extension of H/X_a according to multiplicity.

Furthermore the algorithms we are looking for, in order to build those linear extensions, must be of good algorithmic complexity (if possible linear-time algorithms).

III.3 TWO INHERITANCE ALGORITHMS.

DEFINITION 9 : A depth-first in a graph $G = (X,U)$ yields two total orderings of the vertices, namely : *dfi(G)* (resp. *dfo(G))* which is, for every vertex $x \in X$, defined by $dfi(x) = i$ (resp. $dfo(x) = i$) if x enters (resp. quits) the stack of the depth-first search at the i^{th} rank.

The following theorem relates depth-first search and dfgreedy linear extensions.

THEOREM : BOUCHITTE et al. [1986]. Let P be a poset with a unique minimal element a, and G(a) be the diagram of P (i.e. the graph of the relation P with no transitivity arcs) then there is an isomorphism between the set of dfgreedy linear extensions of P and the set of dfo^d orders yielded by depth-first searches on G(a) starting from the vertex a.

This immediately yields our first inheritance algorithm.

ALGORITHM 1 :

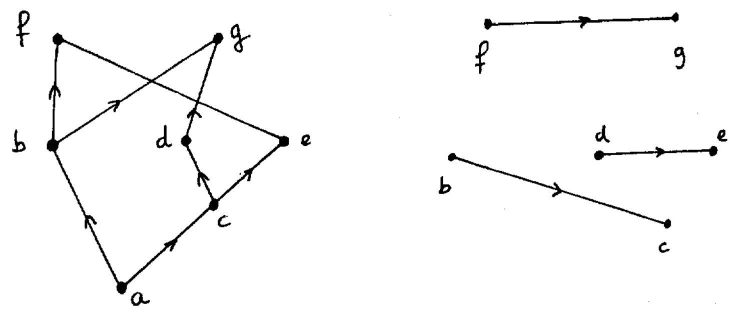
1. Perform a depth-first search of G(a), according to μ^d, starting from the vertex a.
2. $\varphi^1{}_a = dfo^d(G(a))$;

(i.e. in the depth-first search that gives dfo(G(a)) in each vertex b its successors are explored following the decreasing values of the total ordering $\mu(b)$, from right to left in our figures).

PROPOSITION 1 : φ^1 satisfies principle 1 and when G(a) has no transitivity arcs it also satisfies property 2. Furthermore its complexity is in $O(n+m)$ with $n = |X_a|$ and $m = |H/X_a|$.

Proof : From TARJAN [1972], we know that $dfo^d((G(a))$ is a linear extension of H/X_a, and from theorem 1, we know its a dfgreediness. This algorithm is obviously linear, since we only use a depth-first search in the inheritance graph.

Unfortunately, $\wp^1$ does not satisfy principle 4, as can be seen in figure 4.



$\wp^1_a$ gives [a,b,c,d,g,e,f] and the unique linear extension of $H/X_a \cup M_a$ is [a,b,c,d,e,f,g].

Fig. 4 : an example where $\wp^1$ contradicts principle 4.

In the case where multiplicity and inheritance do not contradict themselves, we can obtain a second algorithm using local modifications, which satisfies principle 4. More precisely let us call *contradiction arc* an arc $(x,y) \in M_a$ such that $dfo^d(y) < dfo^d(x)$. (For example (f,g) is a contradiction arc in figure 4).

When performing algorithm 1, we can detect such arcs and then change their status from M_a to H/X_a. Let us call this operation a *setting*. Then we obtain the following algorithm.

ALGORITHM 2 :

```
1. CA ← $M_a$; $M_0$ ← $M_a$;

2. While CA ≠ ∅
   Perform a depth-first search on G(a), according to $\mu^d$, starting from a.
   CA ← {contradiction arcs of this search} ∩ $M_0$;
   If $dfo^d(G(a))$ is a linear extension of $H/X_a \cup M_a$ we have finished;
   Else perform the setting of all contradictions arcs.
   $H/X_a$ ← $H/X_a \cup CA$; Update also $M_a$;

3. $\wp^2_a = dfo^d(G(a))$;
```

PROPOSITION 2 : When there is no contradiction between multiplicity and inheritance, $\wp^2$ yields a dfgreedy linear extension of $H/X_a \cup M_a$. Its complexity is $O(k(n+m))$ where k denotes the number of iterations in the above procedure.

Hints for the proof : Clearly this algorithm gives a linear extension of $H/X_a \cup M_a$. The only problem is the on-line characterization of the contradiction arcs. Happily this can be done by noticing that the example of figure 4, is a characteristic one. This yields the annonced complexity.

Thus we finally have obtained an inheritance search that satisfies principles 1,4, and property 2. Furthermore, we can imagine an algorithm $\wp$ which first perform the poset $H/X_a \cup M_a$ and then apply on it the algorithm 1. But the resulting complexity is analogous to those of transitive closure algorithms. A discussion on this algorithm as well as a study of the contradiction between the multiplicity and inheritance is presented in DUCOURNAU and HABIB [1986].

IV SOME CONCLUSIONS

These two algorithms presented has been implemented in the frame-based language YAFOOL (see DUCOURNAU and QUINQUETON [1985]). The first one is now the standard inheritance algorithm of this language. The second one is much more time consumer and will be only used for special applications that need it.

Furthermore since the great challenge for representing real-world knowledge is to be able to support various exceptions and commonsense reasoning (see CHOURAQUI et al. [1985] or KAYSER [1984]), we have also investigated the problems yielded by multidimensional inheritance as well as that of carrying exceptions in our inheritance algorithms (see DUCOURNAU and HABIB [1987] and DUCOURNAU [1987]). In our framework, we have obtained algorithmic results (generalizations of the algorithms $\wp^1$ and $\wp^2$) on exceptions problems very similar to those studied by ETHERINGTON and REITER [1983], FAHLMAN [1979], FAHLMAN et al. [1981], or TOURETZKY [1986].

In fact, a kind of multidimensional inheritance is used in Le_lisp (CHAILLOUX et al. [1986]), for the primitives SEND2 and GETFN2, and is also used in YAFLOG (DUCOURNAU [1986]) which is a PROLOG-like inference engine written in YAFOOL. So, we are convinced that we have only made the first observations about the following problem : *What are the linear extensions involved in inheritance systems, and how to compute them ?* Much work is still to be done, and it seems to be a very promising research area, for example to determine properties analogous to property 2, which found semantically inheritance and are easy to compute.

V REFERENCES.

1. H. AIT-KACI, R. NASR, *"LOGIN : a logic programming language with built-in inheritance"*, The Journal of Logic Programming, n°3 (1985) 185-215.

2. J. BEZIVIN, *"Langages objets et prototypage"*, Rapport de Recherche N°86-9, Lab. Informatique de Brest, 1986.

3. D.G. BOBROW, K. KAHN, G. KICZALES, L. MASINTER, M. STEFIK, F. ZDYBEL, *"Common loops merging Lisp and object-oriented programming"*, OOPSLA Conf., ACM (1986) 17-29.

4. V. BOUCHITTE, M. HABIB, M. HAMROUN, R. JEGOU, *"Depth-first search and linear extensions"*, Rapport de Recherche N°86-5, Laboratoire Informatique de Brest, 1986.

5. R.J. BRACHMAN, *"What IS-A is and isn't : an analysis of taxonomic link in semantic network"*, Computer, Vol. 16, N°10 (1983) 30-36.

6. J.P. BRIOT, *"Instanciation et héritage dans les langages objets"*, Thèse de $3^{ème}$ cycle, Paris 1985.

7. J. CHAILLOUX et al., *"Le_Lisp, version 15.2"*, INRIA 1986.

8. E. CHOURAQUI, H. FARRENY, D. KAYSER, H. PRADE, *"Modélisation du raisonnement et de la connaissance"*, TSI 4:4 (1985) 391-399.

9. O. COGIS, M. HABIB, *"Nombre de sauts et graphes série-parallèles"*, RAIRO, Informatique Théorique / Theoretical Informatics, vol. 13, n^o1 (1979) 3-18.

10. R. DUCOURNAU, M. HABIB, *"De l'héritage multiple dans les langages orientés objet"*, Working Paper N^o 87-1, Laboratoire Informatique de Brest, janvier 1987, submitted.

11. R. DUCOURNAU, J. QUINQUETON, *"YAFOOL, encore un langage objet à base de frames"*, Rapport Technique N^o72, INRIA, août 1986.

12. R. DUCOURNAU, *"YAFLOG, une implémentation de PROLOG en YAFOOL"*, Rapport interne, SEMA-METRA, novembre 1986.

13. P. DUGERDIL, *"A propos des mécanismes d'héritage dans un langage orienté objet"*, CIIAM 86 Intelligence Artificielle, Marseille (1986) 67-77.

14. D.W. ETHERINGTON, R. REITER, *"On inheritance hierarchies with exceptions"*, Proc. AAAI (1983) 104-108.

15. S.E. FAHLMAN, *"NETL : a system for representing and using real-world knowledge"*, MIT Press, Cambridge, MA, 1979.

16. J. FERBER, *"MERING : un langage d'acteur pour la représentation et la manipulation des connaissances"*, Thèse de Docteur Ingénieur, Université Paris VI, 1983.

17. A. GOLDBERG, D. ROBSON, *"SMALLTALK : the language and its implementation,* Addison-Welsey, 1983.

18. D. KAYSER, *"Examen de diverses méthodes utilisées en représentation des connaissances"*, Actes RFIA 4, tome 2, Paris (1984) 115-144.

19. H. LIEBERMAN, *"Delegation and inheritance : two mechanisms for sharing knowledge in object-oriented systems"*, Journées Langages Orientés Objet, Bigre+Globule, N^o48 (1986) 79-89.

20. B. MEYER, *"EIFFEL, user's manual"*, Software Engineering inc., Santa Barbara, 1986.

21. D.A. MOON, *"Object programming with FLAVORS"*, in Object Oriented Programming Languages Conference, OOPSLA Proceedings, ACM SIGPLAN Notices (1986) 1-8.

22. O. PRETZEL, Problem presented at Oberwolfach Conference on Ordered Sets (1985).

23. R.E. TARJAN, *"Depth-first search and linear graph algorithms"*, SIAM J. of Computing 1:2 (1972) 146-169.

24. D.S TOURETZKY, *"The mathematics of inheritance systems"*, Research Notes in Artifical Intelligence, Pitman 1986.