The Complexity of Late-Binding in Dynamic Object-Oriented Languages*

Enrico Pontelli, Desh Ranjan, Gopal Gupta

Laboratory for Logic, Databases, and Advanced Programming
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003 USA
e-mail: {epontell,dranjan,gupta}@cs.nmsu.edu

Abstract. We study the algorithmic complexity of supporting *late binding* in *dynamic* object-oriented programming (*OOP*) languages—i.e., languages that allow creation of new class definitions at runtime. We propose an abstraction of the late-binding problem for dynamic OOP languages in terms of operations on dynamic trees, and develop a lower bound of $\Omega(\lg n)$ per operation (where n is the number of nodes in the tree). This result shows that efficient solutions (i.e., solutions that incur a constant time overhead per operation) of this problem are, in general, not possible. We also propose new data-structures and algorithms for solving the late-binding problem for dynamic OOP languages very efficiently, with a worst-case time complexity of $O(\sqrt[3]{n})$ per operation. This result is an improvement over most existing approaches.

1 Introduction

We study the the algorithmic complexity of implementation mechanisms for supporting inheritance in *dynamic* object-oriented languages. The aim of this study is to model these implementation mechanisms in terms of operations on dynamic data structures, with the final goal of determining their complexity. Our complexity study is used to derive novel data structures for implementing inheritance mechanisms efficiently. In this work we limit ourselves to studying the complexity of implementing late-binding inheritance in dynamic OOP systems, with particular focus on prototype-based languages. The inheritance considered is single inheritance, or multiple inheritance with linearization. Generalization of this work to more complex domains (tree and graph-based multiple inheritance) is still under investigation.

Inheritance refers to the property of OOP languages whereby new classes are defined by specializing existing classes. When a class D is defined as a subclass of another class B (sub-typing), it inherits all the attributes (both data and operations) from class B. *Late binding* means that procedure-call name-resolution,

* This work has been partially supported by NSF grants HRD 93-53271, INT 95-15256, and CCR 96-25358, and by NATO Grant CRG 921318.

i.e., mapping of a procedure call to a procedure definition, is done at run-time, rather than compile-time. This is because due to sub-typing and inheritance, it is not immediately clear which definition should be used corresponding to a procedure call, as procedure definitions may be multiply defined in a class hierarchy. Late-binding is an essential feature of OOP and is present in most languages, e.g., Java, CLOS, Smalltalk, C++ (virtual functions), etc.

Dynamic OOP languages are languages that allow dynamic runtime creation of class definitions. Thus the complete class hierarchies are not known at compile-time. CLOS [17] and Smalltalk are two such languages. Additionally, prototype based languages [2, 4, 29] are also examples of such languages. Name-resolution in presence of late-binding is easily solved for static OOP languages (such as Java and C++) since the complete class hierarchy is known at compile-time; for dynamic languages, due to absence of this information, it is not as easy. To study the complexity of the problems that arise in implementing name-resolution in late-binding dynamic OOP languages, we first abstract it and formalize it in terms of operations on *dynamic trees* (i.e., trees that grow and shrink with time). This formalization permits us to abstractly study the implementation problems.

The abstraction of the computing machine that we choose is the *Pointer Machine* [18, 19, 27]. The reason we choose a pointer machine is because it provides operations needed for expressing computations over dynamic data structures; it also allows us to perform a finer level of analysis compared to other models (e.g., RAM). Besides, use of pointer machines enables us to use many existing results [22, 12, 14, 21] which are useful in our study and have been developed on pointer machines. Analysis of the name-resolution problem in late-binding dynamic language on pointer machines gives us a deep insight into the sources of its implementation complexity.

Based on our rigorous study, we develop a lower bound time complexity of $\Omega(\lg n)$ per operation (where n is the number of nodes in the tree), for the name-resolution problem in late-binding dynamic OOP languages. This lower bound implies that the implementation of name-resolution (creation, maintenance, member look-up, etc. of a class hierarchy) in late-binding dynamic OOP languages will at least incur cost proportional to $\lg n$, where n is the number of classes in a hierarchy. In particular, this means that *all the operations needed to support dynamic look-up in a class hierarchy cannot be made constant-time.*

We also propose efficient new data-structures and algorithms for solving this problem which have a worst-case time complexity of $O(\sqrt[3]{n})$ per operation. This result is an improvement over existing approaches that typically have a complexity of $\Omega(n)$ per operation. The data structures are very practical and can be easily employed in actual implementations.

The abstraction of the name-resolution problem for dynamic OOP languages in terms of dynamic trees, the derivation of a lower bound on complexity of name-resolution, and the development of an efficient, new implementation technique for it, are the main contributions of this paper. To the best of our knowledge, this is the first rigorous study of implementation of name-resolution in late-binding dynamic object-oriented languages. Our study is inspired by our observation that