Programming Objects with ML-ART
An extension to ML with Abstract and Record Types

Didier Rémy

INRIA-Rocquencourt, BP 105, F-78153 Le Chesnay Cedex.

Abstract. Class-based objects can be programmed directly and efficiently in a simple extension to ML. The representation of objects, based on abstract and record types, allows all usual operations such as multiple inheritance, object returning capability, and message transmission to themselves as well as to their super classes. There is, however, no implicit coercion from objects to corresponding ones of super-classes. A simpler representation of objects without recursion on values is also described. The underlying language extends ML with recursive types, existential and universal types, and mutable extensible records. The language ML-ART is given with a call-by-value semantics for which type soundness is proved.

Introduction

An important motivator for type-checking extensible records is their application to object encoding. Initiated by Cardelli in 1984 [Car84], continued by Wand [Wan87], and then many others, record type-checking has produced several satisfactory solutions for higher order languages [CM89, HP90] and for ML [JM88, OB88, Oho90, Rém93b]. Object encoding, based on record calculi, has revealed severe difficulties, mainly due to over-reliance on recursive values. Consequently, the tendency has been to design languages with objects as primitive operations [Bru93, AC94, Hen91a, MHF93], rather than encodings, to achieve important simplification of type-theoretical models.

Pierce and Turner produced convincing evidence that object-oriented programming can be treated as a matter of programming style, at least from a theoretical point of view [PT93]. However, the use of $F^\omega$ as the base language supports the idea that encodings involve complex type theories, and the demonstration does not always apply to the ML programmer. The need to write many coercions, due to the use of explicit types and to the absence of record extension, makes it obvious that large-scale object-oriented applications cannot be programmed directly in $F^\omega$. Finally, the encoding is created in a call-by-name language, which results in a duplication of too many structures. A recent version of the encoding in a call-by-value language [Pie93] still contains inherent inefficiencies. At least a large amount of syntactic sugar must be provided to program objects in $F^\omega$. 
We concur with the claim that object-oriented programming is essentially a matter of style. Consequently, we do not address it in this paper. Our main goal is to demonstrate that objects can be programmed in a small extension to ML. Therefore, we repeat Pierce’s method utilizing, however, a basic language derived from ML. This results in a quite elegant and still flexible class-based object-oriented programming style, almost as concise as if objects were primitive. No syntactic sugar is required. This approach enables programming capabilities such as multiple inheritance, object returning ability and message transmission to itself as well as its super class. We recognize that implicit coercion of objects to their counterparts in super classes is not possible. This is a serious restriction of our approach to objects.

As in [PT93], we consider objects as abstract data structures, but our encoding differs in two essential ways. First, we can take advantage of record extension to implement inheritance in a simpler way which avoids successive coercions and treats classes as “first class citizens”. Ignoring implicit class coercions enables us to move the recursion on “self” from method vector creation to method application, converting objects to non recursive values.

The second interest of this paper is the language ML-ART utilized for programming objects; it extends core ML with several orthogonal features. None of these is really new, however, the combination is original. We give a complete definition of ML-ART and verify type soundness, but we omit type inference.

The most important feature of ML-ART is extensible records. We choose those described in [Rém93b], although other choices are permissible, provided they implement polymorphic access and polymorphic extension. Polymorphic access refers to the ability to define a function that reads the same field of many records, with different domains. This is the key operation to message passing. Record extension is the operation that creates new records from older ones, via addition of new fields. It is said to be polymorphic if it can operate on records with different domains. Record extension is used to program single inheritance [Wan87] or even multiple inheritance reusing the trick that provides record concatenation through record extension only [Rém93c].

The language is also enriched with recursive types. Record types and recursive types are sufficient to program objects with value abstraction, modulo serious inefficiencies and difficulties with the compilation of recursive values. Thus, we choose to extend the language with existential types, as described by Laufer and Odersky [LO92], and use type abstraction to conceal the internal state of objects. This necessitates replacement of record types with more expressive projective types [Rém92b]. Finally, existential types introduce scope borders which can only be crossed using universal types in a dual way.

The paper is organized as follows. In section 1, we compare encodings of objects with type and value abstraction. In section 2 we informally introduce the language ML-ART. We motivate and introduce its components, one by one. A formal presentation is supplied in the appendices. In section 3, we show how objects can be programmed in ML-ART. In the final section, we discuss and conclude the experience.