Types for Non-Standard Class Operations

Stevie Strickland

@inproceedings{bc-mixins,
    author = {Gilad Bracha and William Cook},
    title = {Mixin-based inheritance},
    booktitle = {OOPSLA/ECOOP ’90: Proceedings of the European conference on Object-oriented programming systems, languages, and applications},
    year = {1990},
    pages = {303--311},
    http://doi.acm.org/10.1145/97945.97982
}

Bracha and Cook illustrate how existing object-oriented languages with only single inheritance, such as Smalltalk and Beta, cannot describe extensions to a family of classes without duplicating either the extension or the base classes. To this end, they examine the idea of mixins from the Common Lisp Object System (CLOS) and show how to adapt the idea of mixins to other object-oriented systems in a principled manner. They provide an example by adding mixins to an existing object-oriented language, Modula-3.

This paper separates the idea of classes that abstract over their superclass from the (implicit) linearization process that is performed in CLOS. That is, it abstracts the policy (abstract subclasses) from the mechanism (linearization). In addition to making the mechanism explicit via mixin application, which decides the order of overriding methods, it also shows that the idea of superclass abstraction is useful for other object-oriented languages.
Flatt et al. show the extension of a Java-like language to include mixins. First they develop CLASSICJAVA, a model of Java, which includes mutable fields and interfaces. They produce both a type system and an operational semantics for the model and prove it sound. Second they describe MIXED-JAVA, an extension of this model, which adds mixins and illustrate how both the type system and operational semantics are affected. Finally, they prove that the resulting system is also sound.

This paper presents one of the first small models for the Java programming language—notably, this model predates Featherweight Java. Also, the paper both provides a full-fledged specification of the mixin mechanism and proves type soundness of the resulting language, which is lacking in earlier work on mixins.
Allen et al. argue for the ability to use generic types at any location where concrete types are allowed, in particular superclass declarations and class instantiation. They show how this leads directly to a natural inclusion of mixins. They also describe the problems that come from naively relaxing the restriction on generic types and how to avoid them.

This paper removes the normal limitations on generic types found in previous models of Java with parameterized classes. Instead, it describes a much smaller limitation to avoid problems such as cyclic class hierarchies and shows that the increased expressivity immediately leads to desired features such as mixins.

Schärdi et al. develop traits, a new method for refactoring common functionality into an abstract form. Traits are bundles of methods, which may be combined in various ways and then applied to a base class. Unlike mixins, where the order of composition affects the result, trait composition is commutative. Also, any conflicts that arise during composition from similarly-named methods must be handled explicitly instead of relying on an implicit system of overriding. They describe an implementation of traits in Squeak, a Smalltalk implementation, and provide an empirical study of their usefulness by refactoring the Smalltalk-80 collection hierarchy.

This paper introduces the idea of traits, which avoids the issue of linearization in mixin application. In addition, it includes an empirical study of the use of traits in refactoring an existing code base, instead of merely asserting the usefulness of the feature.
Fisher and Reppy develop a small calculus that describes the essential flavor of traits and their operations. Unlike the traits of Schärli et al., which dealt only with methods, Fisher and Reppy allow traits to require that the base class to which they are applied must define certain fields. Their calculus includes both a type system and an operational semantics, and they sketch a soundness proof.

This paper is the first paper to take the idea of traits and shows that traits can be added to a typed object-oriented language in a sound manner.