Towards a Semantics for Object-Z

David Duke  Roger Duke

Key Centre for Software Technology
Department of Computer Science
University of Queensland

Abstract

Object-Z is a notation based on Z but with extensions to more fully support an object-oriented style of specification. Object-Z uses the class concept to incorporate a description of an object’s state with related operations. In this paper we introduce an underlying set-theoretic model for classes and so give a formal semantics for classes which extends the semantics of schemas in Z. Our model for a class is based on the idea of a history which captures the sequence of operations and state changes undergone by an instance (object) of the class. Part of the specification of a class can involve predicates which restrict the possible histories of an object.

1 Introduction

Recent collaborative work with the Overseas Telecommunication Corporation (OTC) of Australia has involved the specification in Z of complex software systems requiring many state and operation schemas. Although Z is an excellent notation in which to express such specifications, it lacks both appropriate high level structuring, and a design strategy to handle large systems.

One approach to tackling this problem of complexity in large systems has been to develop Object-Z, an object-oriented extension of Z. Object-Z is currently being used commercially to produce substantial specifications in non-trivial application areas [22].

Object orientation is a concept that has gained rapid popularity in the programming language community, but, perhaps because of its lack of a formal basis, has not attracted so much attention from the specification field. The aim of this paper is not to put forward a proposal for a formal model of object orientation, but rather to show how object-oriented concepts can be introduced into Z without compromising formality.

Object-Z is based on the idea of a class which encapsulates into one construct related state and operation schemas. A consequence of this structuring is that both states and operations can be inherited within other classes, facilitating the incremental construction of large specifications [28] and leading to further structuring of the overall specification. Within the extended framework, other object-oriented concepts such as subtyping can then be introduced, providing further benefits to the specifier.
An instance of a class is called an object; classes and objects are related just as types and instances in Z. An object may be declared in an Object-Z specification just as a variable of any type may be declared in Z.

It is important to note that this paper does not purport to be the full semantics for Object-Z. Instead, our aim is to present the core ideas that underly the extensions, and show how they may be incorporated into the existing set-theoretic semantics of schemas as specified by Spivey [26]. Work is still underway to identify and define constructs such as class operators, which work analogously to the schema operators $\wedge, \vee, \gg$, etc.

Our formal model is based upon the idea of a class history; a history of an object is a sequential record of the operations, together with the underlying operation states, undergone by that object. A class is modelled by the set of all possible histories that an object of that class can undergo. By a simple extension of the Z type framework, it is possible to treat classes as types, and hence consider the declaration of variables of a class type.

2 An Introduction to Object-Z Classes

In this paper we give only a brief overview to Object-Z. For a more complete introduction together with illustrative case studies see Carrington et al [6].

The syntactic structure of a class is as follows:

```
ClassName
  inherited classes
  constants
    variables
      state invariant
  initial state
  operations
    history invariant
```

As an illustrative example, consider the following specification of the class `Queue` of a sequence of (unspecified) messages. This class has a constant, `Maxq`; one variable, `items`; an initial state, specified in the schema `Init`; and two operations, `Join` and `Leave`. Operations are often referred to as `methods` in the literature on object orientation.