Small Oz is expressive enough to support the object-oriented abstractions introduced in the previous two chapters. This comes of little surprise since Small Oz subsumes functional programming and the presented object system is sufficiently close to existing object systems for functional languages like Flavors [Moo86] and CLOS, which get by with little or no semantic extension of the base language Lisp. A secondary issue is then how the syntactic sugar for classes that we could not resist introducing can be translated to plain Small Oz.

The object-oriented abstractions are provided by a Small Oz program, called object library. In this chapter, we will sketch this library and the syntactic reduction of the class syntax to Small Oz. These two components together can be seen as a simple semantic foundation for Objects in Oz.

The library must be constructed in such a way that the safety conditions introduced in the previous two chapters are met. In particular, programs that use the library must be protected in the following way.

- Attributes must not be accessible from outside an object,
- private attributes, features, and methods must not be accessible outside their class definition, and
- insecure multiple inheritance must be prevented.

The user is free to define his own object-oriented abstractions, and—as we have seen in Chapter 4—Oz provides a wide variety of possibilities in this respect. The point is that these abstractions should not be allowed to mingle with code that uses the object system. For example, user programs should not allow to unsafely manipulate classes and objects provided by standard libraries such as the window system.

### 7.1 Class Definition

The obvious approach to reduce class definition to Small Oz is to translate it to the application of a fixed procedure. The arguments of this procedure can describe the parents, attributes, features, properties and methods of the class to be defined. For example, the class definition
Program 7.1 Overall Structure of Object Library

```
declare MakeClass New ObjectApply MethodApply
        AttrAssign AttrAccess AttrExchange in
local
    OODesc={NewName} OOAncestors={NewName} ...
in
    fun {MakeClass ...} ... end
    fun {New ...} ... end
    proc {ObjectApply ...} ... end
    proc {MethodApply ...} ... end
    fun {AttrAssign ...} ... end
    proc {AttrAccess ...} ... end
    fun {AttrExchange ...} ... end
end
```

class Account
    from BaseObject
    prop final
    attr balance:O
    feat fee:2
    meth transaction(Amount) ... end
    meth getBalance(B) ... end
end

is translated to the following application of the procedure MakeClass. 1

```
Account={MakeClass
desc(parents: [BaseObject]
        properties: [final]
        attributes: [balance#O]
        features: [fee#2]
        methods: [transaction#proc {$ ...} ... end
                        getBalance #proc {$ ...} ... end])
```

The procedure MakeClass defines how classes are represented in Small Oz. The definitions of the procedure MakeClass together with New for object creation form the core of the object library. We will introduce other procedures that define object and method application (ObjectApply, MethodApply) and attribute manipulation (AttrAccess, AttrAssign, AttrExchange).

The record desc(⋯) defines all properties of the class, and thus we could use this record to represent the class. However, in order to meet the safety conditions, we wrap the class description in another record whose features are names. These names are bound to the local variables OODesc and OOAncestors whose scope is limited to the object library. Program 7.1 shows the overall structure of the object library.

Program 7.2 shows the definition of the procedure MakeClass. The only fact that is known to the user about a class C defined by MakeClass is that it is a record with

1 In Oz, a syntactic convention makes sure that the user cannot accidentally redefine such implicitly used variables and thus render parts of the object system unusable.