Aslan Oztreves

CS-360

Krzysztof Nowak

Summary 7.1 – 7.4

**7.1 - Overview**

Computer hardware can interpret bits in memory in several different ways: as instructions, addresses, characters, and integer and floating-point numbers of various lengths. The bits themselves, however, are untyped: the hardware on most machines makes no attempt to keep track of which interpretations correspond to which locations in memory. Assembly languages reflect this lack of typing: operations of any kind can be applied to values in arbitrary locations. High-level languages, by contrast, almost always associate types with values, to provide the contextual information and error checking alluded to above.

* 7.1.1 The Meaning of “Type”
* 7.1.2 Polymorphism
* 7.1.3 Orthogonality
* 7.1.4 Classification of Types

**7.2 - Type Checking**

In most statically typed languages, every definition of an object (constant, variable, subroutine, etc.) must specify the object’s type. Moreover, many of the con- texts in which an object might appear are also typed, in the sense that the rules of the language constrain the types that an object in that context may validly possess. In the subsections below we will consider the topics of *type equivalence*, *type compatibility*, and *type inference*. Of the three, type compatibility is the one of most concern to programmers. It determines when an object of a certain type can be used in a certain context.

* 7.2.1 Type Equivalence
* 7.2.2 Type Compatibility
* 7.2.3 Type Inference
* 7.2.4 Type Checking in ML

**7.3 - Parametric Polymorphism**

As we have seen in the previous section, functions in ML-family languages are naturally polymorphic. Consider the simple task of finding the minimum of two values.

* 7.3.1 Generic Subroutines and Classes
* 7.3.2 Generics in C++, Java, and C#

**7.4 - Equality Testing and Assignment**

For simple, primitive data types such as integers, floating-point numbers, or characters, equality testing and assignment are relatively straightforward operations, with obvious semantics and obvious implementations (bit-wise comparison or copy). For more complicated or abstract data types, both semantic and implementation subtleties arise.