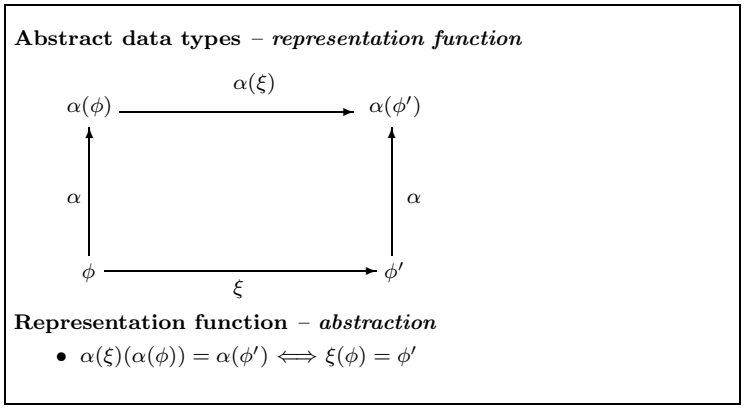
0.1 Objects as behavioral types

A syntax-directed correctness calculus as presented in section **??** provides, in principle, excellent support for a problem-oriented approach to program development, provided that the requirements a program has to meet can be made explicit in a mathematical, logical framework.

When specifying requirements, we are primarily interested in the abstract properties of a program, as may be expressed in some mathematical domain. However, when actually implementing the program (and verifying its correctness) we mostly need to take recourse to details we do not wish to bother with when reasoning on an abstract level. In this section we will discuss how we may verify that an abstract type is correctly implemented by a behavioral (implementation) subtype, following [Am90]. Also, we will define precise guidelines for determining whether two (behavioral) types satisfy the (behavioral) subtype relation, following [Liskov93].

0.1.1 Abstraction and representation

In [Am90] a proposal is sketched to define the functionality of objects by means of *behavioral types*. Behavioral types characterize the behavioral properties of objects in terms of (possible) modifications of an abstract state. So as to be able to ignore the details of an implementation when reasoning about the properties of a particular program, we may employ a *representation abstraction function* which maps the concrete data structures and operations to their counterparts in the abstract domain.



0-1

Slide 0-1: Abstraction and representation

The diagram in slide 0-1 pictures the reasoning involved in proving that a particular implementation is correct with respect to a specification in some abstract mathematical domain. Assume that we have, in the concrete domain, an action a that corresponds with a state transformation function ξ. Now assume