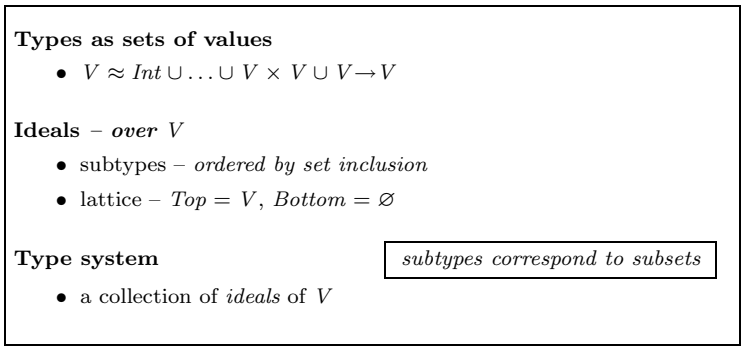
0.1 The subtype relation

In this section, we will study the subtype relation in a more formal manner. First we investigate the notion of subtypes in relation to the interpretation of *types as sets*, and then we characterize the subtype relation for a number of constructs occurring in programming languages (such as ranges, functions and records). Finally, we will characterize objects as records and correspondingly define the subtype relation for simple record (object) types. These characterizations may be regarded as a preliminary to the type calculi to be developed in subsequent sections.

0.1.1 Types as sets

A type, basically, denotes a set of elements. A type may be defined either extensionally, by listing all the elements constituting the type, or descriptively, as a constraint that must be satisfied by an individual to be classified as an element of the type.



Slide 0-1: The interpretation of types as sets

Formally, we may define the value set of a type with subtypes as an isomorphism of the form

$$V \approx Int \cup \ldots \cup V \times V \cup V \rightarrow V$$

which expresses that the collection of values V consists of (the union of) basic types (such as *Int*) and compound types (of which V itself may be a component) such as record types (denoted by the product $V \times V$) and function types (being part of the function space $V \rightarrow V$).

Within this value space V, subtypes correspond to subsets that are ordered by set inclusion. Technically, the subsets corresponding to the subtypes must be *ideals*, which comes down to the requirement that any two types have a maximal type containing both (in the set inclusion sense).

Intuitively, the subtype relation may be characterized as a refinement relation, constraining the set of individuals belonging to a type. The subtype refinement