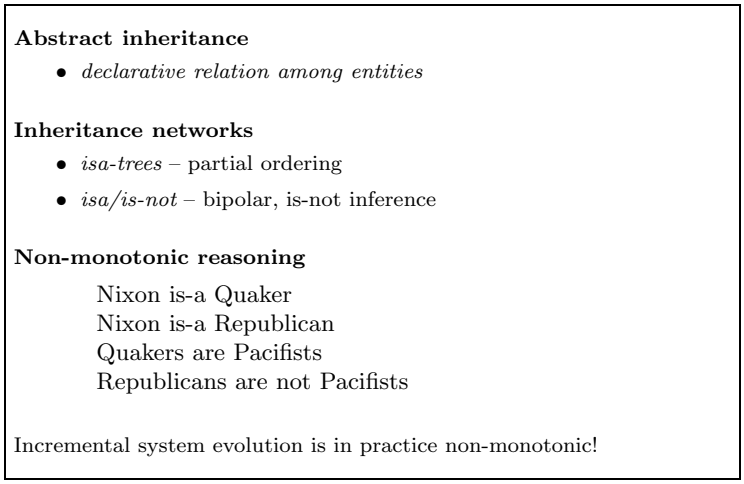
0.1 Abstract inheritance

Inheritance hierarchies play a role both in knowledge representation systems and object-oriented programming languages, see [LNS90].

In effect, historically, the notions of *frames* and *is-a hierarchies* (that play a role in knowledge representation) and the notions of *classes* and *inheritance* (that have primarily been developed in a programming language context) have mutually influenced each other.

In object-oriented programming languages, classes and inheritance are strongly related to types and polymorphism, and directed towards the construction of reliable programming artifacts. In contrast, the goal of knowledge representation is to develop a semantically consistent description of some real world domain, which allows us to reason about the properties of the elements in that domain.



Slide 0-1: Knowledge representation and inheritance

One of the first formal analyses of the declarative aspects of inheritance systems was given in [To86]. The theoretical framework developed in [To86] covers the inheritance formalisms found in frame systems such as FRL, KRL, KLONE and NETL, but also to some extent the inheritance mechanisms of Simula, Smalltalk, Flavors and Loops. The focus of [To86], however, is to develop a formal theory of inheritance networks including defaults and exceptions. The values of attributes play a far more important role in such networks than in a programming context. In particular, to determine whether the relationships expressed in an inheritance graph are consistent, we must be able to reason about the values of these attributes. In contrast, the use of inheritance in programming languages is primarily focused on sharing instance variables and overriding (virtual) member functions, and is not so much concerned with the actual values of instance variables.

Inheritance networks in knowledge representation systems are often *non mono-*

tonic as a result of having *is-not* relations in addition to *is-a* relations and also because properties (for example *can-fly*) can be deleted.

Monotonicity is basically the requirement that all properties are preserved, which is the case for strict inheritance satisfying the substitution principle. It is a requirement that should be adhered to at the risk of jeopardizing the integrity of the system. Nevertheless, strict inheritance may be regarded as too inflexible to express real world properties in a knowledge representation system. The meaning of *is-a* and *is-not* relations in a knowledge representation inheritance graph may equivalently be expressed as predicate logic statements. For example, the statements

- $\forall\, x.Quaker(x) \rightarrow Human(x)$
- $\forall\, x.Republican(x) \rightarrow Human(x)$

express the relation between, respectively, the predicates *Quaker* and *Republican* to the predicate *Human* in the graph above. In addition, the statements

- $\forall\, x.Quaker(x) \rightarrow Pacifist(x)$
- $\forall\, x.Republican(x) \rightarrow \neg Pacifist(x)$

introduce the predicate *Pacifist* that leads to an inconsistency when considering the statement that *Nixon is a Quaker and a Republican*.

Some other examples of statements expressing relations between entities in a taxonomic structure are given in slide **??**.

Taxonomic structure

$\forall\, x.Elephant(x) \rightarrow Mammal(x)$
$\forall\, x.Elephant(x) \rightarrow color(x) = gray$
$\forall\, x.Penguin(x) \rightarrow Bird(x) \wedge \neg CanFly(x)$

0-2

Slide 0-2: Taxonomies and predicate logic

The latter is often used as an example of non-monotonicity that may occur when using defaults (in this case the assumption that *all birds can fly*).

The mathematical semantics for declarative taxonomic hierarchies, as given in [To86], are based on the notion of constructible lattices of predicates, expressing a partial order between the predicates involved in a taxonomy (such as, for example, *Quaker* and *Human*). A substantial part of the analysis presented in [To86], however, is concerned with employing the graph representation of inheritance structures to improve on the efficiency of reasoning about the entities populating the graph. In the presence of multiple inheritance and non-monotonicity due to exceptions and defaults, care must be taken to follow the right path through the inheritance graph when searching for the value of a particular

attribute. Operationally, the solution presented by [To86] involves an ordering of inference paths (working upwards) according to the number of intermediate nodes. Intuitively, this corresponds to the distance between the node using an attribute and the node defining the value of the attribute. In strictly monotonic situations such a measure plays no role, however!