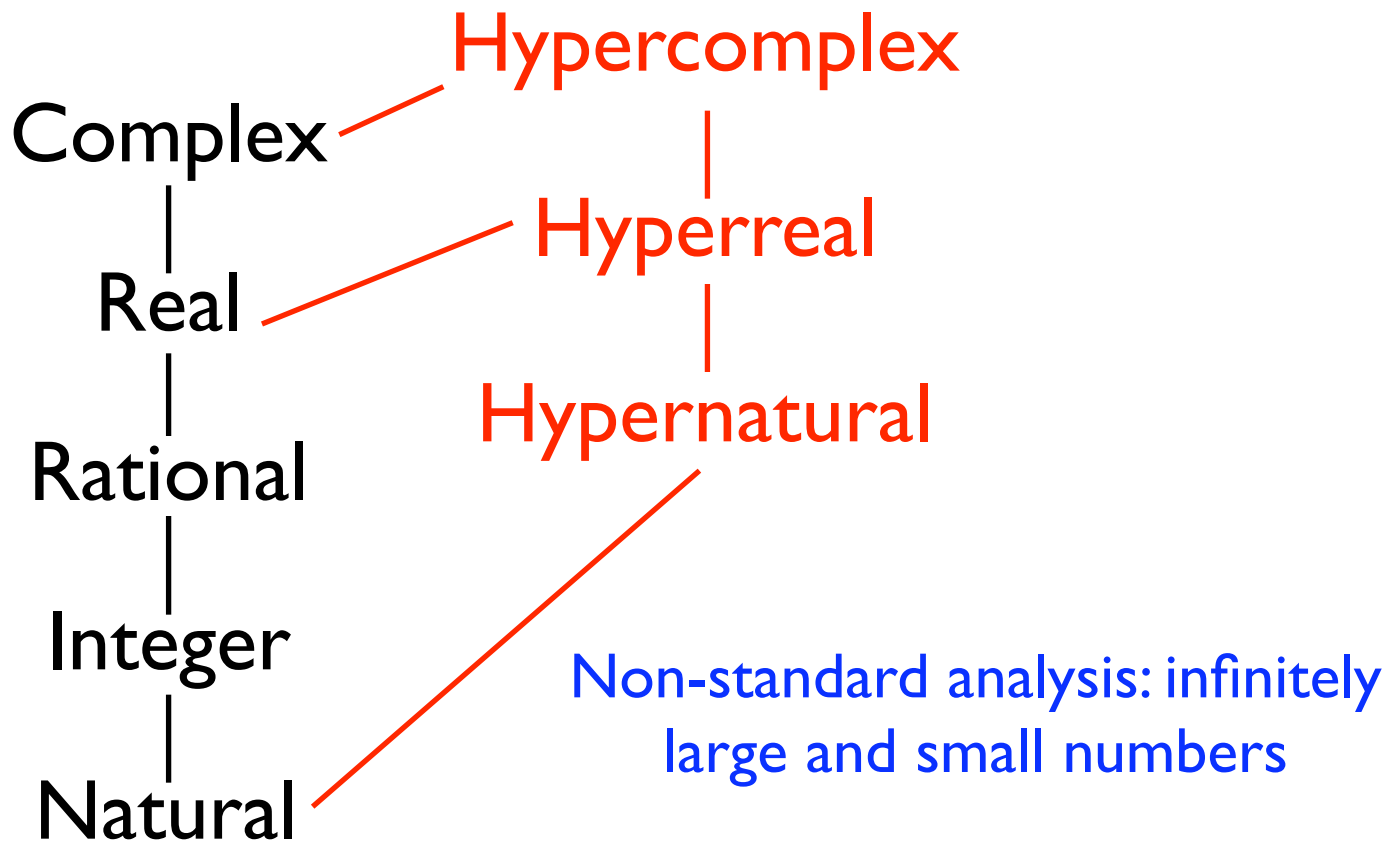
Organizing Numerical Theories using Axiomatic Type Classes

Lawrence C Paulson
Computer Laboratory

UNIVERSITY OF
CAMBRIDGE

Many Kinds of Numbers



Many Arithmetic Laws

- commutative and associative
- distributive and cancellation
- monotonicity and sign-related
- for + − × / abs and exponentiation

There are 100s of laws, and special-purpose code.
Must it be replicated?

Subtyping: The Usual Fix

- Inheritance hierarchy based on inclusions such as nat ⊆ int ⊆ rat ⊆ real ⊆ complex
- Inverts the natural order of construction: the complex numbers actually derive their properties from the reals!
- The complexes are unordered, so laws about < must be inherited from the reals
- New theories (such as polynomials) don't benefit, since they aren't subtypes of anything

Axiomatic Type Classes

- Controlled overloading based on axioms
- Can define concept hierarchies abstractly
- Prove theorems about a concept from its axioms
- Prove that a type belongs to a class, making those theorems available
- Due to Nipkow (1991) and Wenzel (1997)

Defining Semirings

axclass $semiring \subseteq zero,\ one,\ plus,\ times$

$add\text{-}assoc\colon (a + b) + c = a + (b + c)$

$add\text{-}commute\colon a + b = b + a$

$add\text{-}0\ [\quad]\colon 0 + a = a$

$add\text{-}left\text{-}imp\text{-}eq\colon a + b = a + c \Longrightarrow b{=}c$

$mult\text{-}assoc\colon (a * b) * c = a * (b * c)$

$mult\text{-}commute\colon a * b = b * a$

$mult\text{-}1\ [\quad]\colon 1 * a = a$

$left\text{-}distrib\colon (a + b) * c = a * c + b * c$

$zero\text{-}neq\text{-}one\ [\quad]\colon 0 \neq 1$

Ordered Semirings

Existing class of linear orders

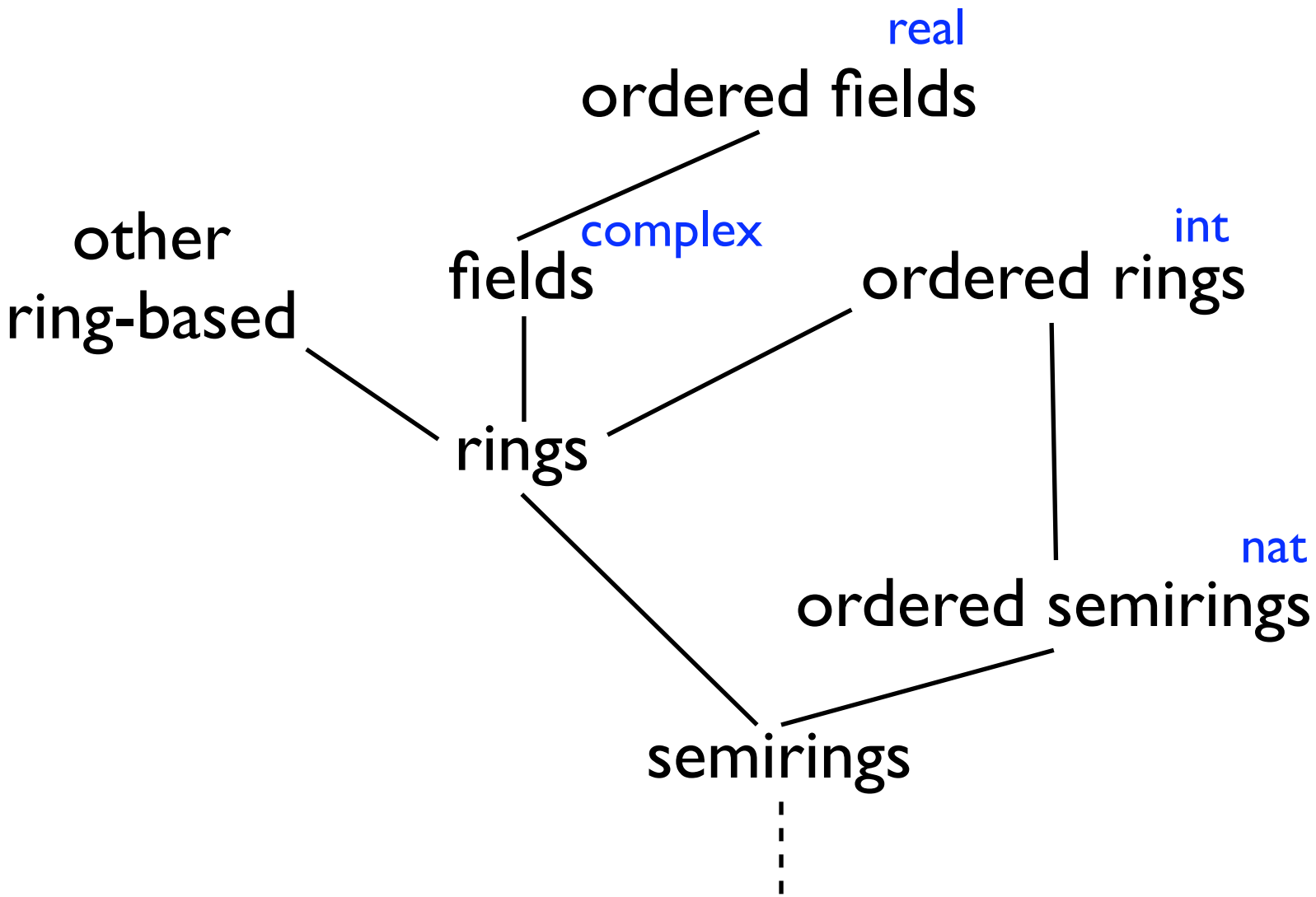
axclass $ordered\text{-}semiring \subseteq semiring, linorder$

$zero\text{-}less\text{-}one\ [simp]\colon\ 0 < 1$

$add\text{-}left\text{-}mono\colon\ a \leq b \Longrightarrow c + a \leq c + b$

- Addition is cancellative and monotonic
- Multiplication distributes over addition
- *Example*: the natural numbers

The Full Hierarchy



The Natural Numbers form a Semiring

instance $nat :: semiring$
proof
  fix $i\ j\ k :: nat$
  show $(i + j) + k = i + (j + k)$ **by** $(rule\ nat\text{-}add\text{-}assoc)$
  show $i + j = j + i$ **by** $(rule\ nat\text{-}add\text{-}commute)$
  show $0 + i = i$ **by** $simp$
  show $(i * j) * k = i * (j * k)$ **by** $(rule\ nat\text{-}mult\text{-}assoc)$
  show $i * j = j * i$ **by** $(rule\ nat\text{-}mult\text{-}commute)$
  show $1 * i = i$ **by** $simp$
  show $(i + j) * k = i * k + j * k$ **by** $(simp\ add{:}\ add\text{-}mult\text{-}distrib)$
  show $0 \neq (1{::}nat)$ **by** $simp$
  assume $k{+}i = k{+}j$ **thus** $i{=}j$ **by** $simp$
qed

And They Form An Ordered Semiring

instance nat :: $ordered\text{-}semiring$
proof
  fix $i\ j\ k$ :: nat
  show $0 < (1::nat)$ **by** $simp$
  show $i \leq j \Longrightarrow k + i \leq k + j$ **by** $simp$
  show $i < j \Longrightarrow 0 < k \Longrightarrow k * i < k * j$ **by ...**
qed

As the type already belongs to class *semiring*, only the additional axioms must be proved.

A Type Class for Powers

$$\begin{array}{l}\textbf{axclass}\ ringpower \subseteq semiring, power \\ \quad power\text{-}0\ [simp]: \quad a\ \hat{}\ 0 \qquad = 1 \\ \quad power\text{-}Suc: a\ \hat{}\ (Suc\ n) = a * (a\ \hat{}\ n)\end{array}$$

- The usual laws follow from these axioms
- Prove them once; use them for each type
- Other common operators can be dealt with in the same way

Setting up Powers for the Naturals

primrec $(power)$
  $p \ \hat{} \ 0 = 1$
  $p \ \hat{} \ (Suc \ n) = (p::nat) * (p \ \hat{} \ n)$

instance $nat :: ringpower$
proof
  fix $z :: nat$
  fix $n :: nat$
  show $z\hat{}0 = 1$ **by** $simp$
  show $z\hat{}(Suc \ n) = z * (z\hat{}n)$ **by** $simp$
qed

Numeric Literals

- Coded as 2's-complement binary numbers
- Valuation defined by primitive recursion
- Correspondence between binary arithmetic and numerical arithmetic proved for rings
- Can be instantiated for all numeric types save the naturals

Uniform Simplification

- Axioms/theorems declared with *[simp]* are used to simplify terms of any suitable type
- Thus simplification is uniform for all the numeric types
- Simplification procedures (HOL *conversions*) also behave uniformly

Summary/Conclusions

- Type classes cope with many numeric types.
- Properties are proved abstractly
- 100s of lemmas become available to a new numeric type
- No need to repeat proofs or code or to invent systematic naming conventions
- Related work: PVS theories?