NICTA

COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Toby Murray, June Andronick, Gerwin Klein

type classes & locales

Slide 1

NICTA

Content

[a] a1 due; [b] a2 due; [c] a3 due

Slide 2

NICTA

Type Classes

Common pattern in Mathematics:

➜ Define abstract structures (semigroup, group, ring, field, etc)
➜ Study and derive properties in these structures
➜ Instantiate to concrete structure: (nats with + and * from a ring)
➜ Can use all abstract laws for concrete structure

Type classes in functional languages:

➜ Declare a set of functions with signatures (e.g. plus, zero)
➜ give them a name (e.g. c)
➜ Have syntax 'a :: c for: type 'a supports the operations of c
➜ Can write abstract polymorphic functions that use plus and zero
➜ Can instantiate specific types like nat to c

Isabelle supports both.

Slide 3

NICTA

Type Class Example

Example:

class semigroup =
fixes mult :: 'a ⇒ 'a ⇒ 'a (**infix** · 70)
assumes assoc: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

Declares:

➜ a name (semigroup)
➜ a set of operations (fixes mult)
➜ a set of properties/axioms (assumes assoc)

Slide 4

Type Class Use

Can constrain type variables 'a with a class:

definition sq :: ('a :: semigroup) ⇒ 'a **where** sq x ≡ x · x

More than one constraint allowed. Sets of class constraints are called **sort**.

Can reason abstractly:

lemma "sq x · sq x = x · x · x · x"

Can instantiate:

instantiation nat :: semigroup
begin
definition "(x::nat) · y = x * y"
instance $< proof >$
end

Slide 5

DEMO: TYPE CLASSES

Slide 6

Type constructors

Basic type instantiation is a special case.

In general:
Type constructors can be seen as functions from classes to classes.

Example:
product type prod :: (semigroup, semigroup) semigroup
(or: pairs of semigroup elements again form a semigroup)

Declarations such as *(semigroup, semigroup) semigroup* are called **arities**.

Fully integrated with automatic type inference.

Slide 7

Subclasses

Type classes can be extended:

class rmonoid = semigroup +
fixes one :: 'a
assumes x · one = x

rmonoid is a **subclass** of semigroup

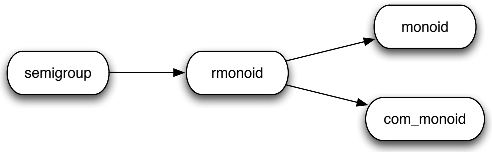
Has all operations & assumptions of semigroup + additional ones.

Can build hierarchies of abstract structures.

Slide 8

More Subclasses

Example structure:



Can prove: every com_monoid is also a monoid.

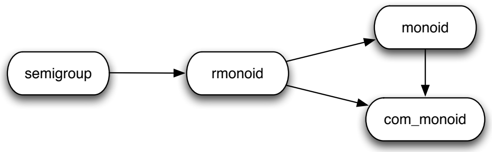
Can tell Isabelle that connection:

subclass (in com_monoid) monoid $< proof >$

Slide 9

Result

Result:



Slide 10

Limitations

Operations (fixes) are implemented by overloading

- ➜ each type constructor can implement each operation only once
- ➜ semigroup must be instantiated to addition or multiplication, not both

Type inference must remain automatic, with unique most general types

- ➜ type classes can mention only one type variable
- ➜ type constructor arities must be co-regular:

$K :: (c_1, ..., c_n)c$ and $K :: (c'_1, ..., c'_n)c'$ and $c \subseteq c'$ $\Longrightarrow$ $\forall i.\ c_i \subseteq c'_i$

Slide 11

DEMO: SUBCLASSES

Slide 12

From Types to Logic

Type classes use the type system to store facts.

lemma	**lemma**
fixes $x :: \alpha ::$ rmonoid	**fixes** $x :: \alpha$
	assumes OFCLASS$(\alpha,$ rmonoid$)$
shows $x \cdot one \cdot y = c \cdot y$	**shows** $x \cdot one \cdot y = c \cdot y$

The type system allows us to manage type assertions **implicitly**.

What if we could implicitly manage a **lemma**? We get **locales**.

Slide 13

Declaring Locales

Declaring **locale** (named context) loc:

locale loc =

$loc1$ +	Import other locales
fixes ...	variables
assumes ...	facts

The **fixes** and **assumes** taken together are called context elements.

Slide 14

Declaring Locales

Theorems may be stated relative to a named locale.

lemma (**in** loc) P [simp]: $proposition$
  $proof$

or

context loc **begin**
lemma P [simp]: $proposition$
  $proof$
end

- ➜ Adds theorem P to context loc.
- ➜ Theorem P is in the simpset in context loc.
- ➜ Exported theorem $loc.P$ visible in the entire theory.

Slide 15

Isar Is Based On Contexts

Structured proofs (Isar) have some similar properties to locales.

theorem $\bigwedge x.\ A \Longrightarrow C$
proof -
  fix x
  assume Ass: A
  ⋮
  from Ass **show** C ...
qed

x and Ass are visible inside this context

Slide 16

Beyond Isar Contexts

NICTA

Locales are extended contexts, look similar to type classes

- ➜ Locales are **named**
- ➜ Fixed variables may have **syntax**
- ➜ Locale may be entered and exited repeatedly
- ➜ It is possible to **add** and **export** theorems
- ➜ It is possible to **instantiate** locales
- ➜ Locale expression: **combine** and **modify** locales
- ➜ No limitation on type variables
- ➜ Term level, not type level: no automatic inference

Slide 17

Context Elements

NICTA

Locales consist of **context elements**.

fixes	Parameter, with syntax
assumes	Assumption
defines	Definition
notes	Record a theorem

Slide 18

NICTA

DEMO: LOCALES 1

Slide 19

Parameters Must Be Consistent!

NICTA

- ➜ Parameters in **fixes** are distinct.
- ➜ Free variables in **defines** occur in preceding **fixes**.
- ➜ Defined parameters cannot occur in preceding **assumes** nor **defines**.

Slide 20

Locale Expressions

Locale name: n

Rename: $n: \ e \, q_1 \dots q_n$

Change names of parameters in e,

Give new locale the name prefix n (optional)

Merge: $e_1 + e_2$

Context elements of e_1, then e_2.

Slide 21

DEMO: LOCALES 2

Slide 22

Normal Form of Locale Expressions

Locale expressions are converted to flattened lists of locale names.

➜ With full parameter lists

➜ **Duplicates removed**

Allows for **multiple inheritance**!

Slide 23

Instantiation

Move from **abstract** to **concrete**.

interpretation label: loc "parameter 1" . . . "parameter n"

➜ Instantiates locale **loc** with provided parameters.

➜ Imports all theorems of **loc** into current context.

- Instantiates theorems with provided parameters.
- Interprets attributes of theorems.
- Prefixes theorem names with **label**

➜ version for local Isar proof: **interpret**

Slide 24

Sublocales

Similar to type classes:

sublocale (in sub_loc) parent_loc $< proof >$

makes facts of parent_loc available in sub_loc.

Slide 25

DEMO: LOCALES 3

Slide 26

We have seen today ...

- ➜ Type Classes + Instantiation
- ➜ Locale Declarations + Theorems in Locales
- ➜ Locale Expressions + Inheritance
- ➜ Locale Instantiation

Slide 27