COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

type classes & locales
Last Time

- more C verification
- preventing undefined execution
- finite machine words
- concrete C data types
- C memory model and pointers
Content

➜ Intro & motivation, getting started

➜ Foundations & Principles
  • Lambda Calculus, natural deduction
  • Higher Order Logic
  • Term rewriting

➜ Proof & Specification Techniques
  • Isar
  • Inductively defined sets, rule induction
  • Datatypes, recursion, induction
  • Calculational reasoning, mathematics style proofs
  • Hoare logic, proofs about programs

Rough timeline

[1] a1 out; b a1 due; c a2 out; d a2 due; e session break; f a3 out; g a3 due
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.
Example:

```haskell
class semigroup =
    fixes mult :: 'a ⇒ 'a ⇒ 'a (infix · 70)
    assumes assoc: (x · y) · z = x · (y · z)
```

Declares:

- a name (semigroup)
- a set of operations (fixes mult)
- a set of properties/axioms (assumes assoc)
Can constrain type variables ’a with a class:

\textbf{definition} \texttt{sq :: (’a :: semigroup) \Rightarrow ’a where sq x \equiv x \cdot x}

More than one constraint allowed. Sets of class constraints are called \textbf{sort}.

Can reason abstractly:

\textbf{lemma} “sq x \cdot sq x = x \cdot x \cdot x \cdot x”

Can instantiate:

\textbf{instantiation} \texttt{nat :: semigroup}

\begin{verbatim}
begin
  \textbf{definition} “(x::nat) \cdot y = x \ast y”
  \textbf{instance} < proof >
end
\end{verbatim}

DEMO: TYPE CLASSES
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 \((\text{semigroup, semigroup}) \text{ semigroup}\) are called **arities**.

**Fully integrated with automatic type inference.**
Subclasses

Type classes can be extended:

```plaintext
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.
More Subclasses

Example structure:

Can prove: every com_monoid is also a monoid.

Can tell Isabelle that connection:

subclass (in com_monoid) monoid < proof >
Result:

- `semigroup` → `rmonoid`
- `rmonoid` → `monoid` → `com_monoid`
Limitations

Operations (fixes) are implemented by overloading

- each type constructor can implement each operation only once

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:
  \[ K :: (c_1, ..., c_n)c \quad \text{and} \quad K :: (c'_1, ..., c'_n)c' \quad \text{and} \quad c \subseteq c' \implies \forall i. \ c_i \subseteq c'_i \]
DEMO: SUBCLASSES
Isar Is Based On Contexts

\textbf{theorem} $\forall x. A \implies C$

\textbf{proof -}

\textbf{fix} $x$

\textbf{assume} $Ass: A$

\textbf{from} $Ass$ \textbf{show} $C$ \ldots \hspace{1cm} x \text{ and } Ass \text{ are visible inside this context}

\textbf{qed}
Locales are extended contexts, look similar to type classes

- Locales are **named**
- Fixed variables may have **syntax**
- 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
Context Elements

Locales consist of **context elements**.

- **fixes** Parameter, with syntax
- **assumes** Assumption
- **defines** Definition
- **notes** Record a theorem
Declaring locales

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

```latex
locale $loc$ =
  $loc_1$ + \text{Import}
  \text{fixes} \ldots \text{Context elements}
  \text{assumes} \ldots
```
Theorems may be stated relative to a named locale.

\textbf{lemma} (in} \textit{loc}) \( P \) [simp]: \textit{proposition}

\begin{proof}
\end{proof}

or

\textbf{context} \textit{loc} begin
\textbf{lemma} \textit{P} [simp]: \textit{proposition}

\begin{proof}
\end{proof}
\textbf{end}

\rightarrow \text{Adds theorem} \textit{P} \text{ to context} \textit{loc}.

\rightarrow \text{Theorem} \textit{P} \text{ is in the simpset in context} \textit{loc}.

\rightarrow \text{Exported theorem} \textit{loc}{}.\textit{P} \text{ visible in the entire theory.}
DEMO: LOCALES 1
Parameters Must Be Consistent!

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

Locale name: \( n \)

Rename: \( n : e q_1 \ldots q_n \)
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 \( e_2 \).
DEMO: LOCALES 2
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!
Instantiation

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

**interpretation** label: loc "parameter 1" \ldots "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: **interpet**
Sublocales

Similar to type classes:

```
sublocale (in sub_loc) parent_loc  < proof >
```

makes facts of parent_loc available in sub_loc.
Demo: Locales 3
We have seen today ...

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