Automated Reasoning

Lecture 7: Locales in Isabelle/HOL

Jacques Fleuriot
jdf@inf.ed.ac.uk

Axiomatic Extensions Considered Harmful

As we saw already, **definitional** extension is favoured over **axiomatic** extension in Isabelle/HOL.

Axiomatic Extensions Considered Harmful

As we saw already, **definitional** extension is favoured over **axiomatic** extension in Isabelle/HOL.

- Axiomatization can introduce an inconsistency.

Axiomatic Extensions Considered Harmful

As we saw already, **definitional** extension is favoured over **axiomatic** extension in Isabelle/HOL.

- Axiomatization can introduce an inconsistency.
- Example: After declaring the existence of a new type *SET* in Isabelle, it is possible to add a new axiom:

axiomatization
$\quad Member :: SET \Rightarrow SET \Rightarrow bool$
where
$\quad comprehension : \exists y. \forall x.\ Member\ x\ y \longleftrightarrow P\ x$

Axiomatic Extensions Considered Harmful

As we saw already, **definitional** extension is favoured over **axiomatic** extension in Isabelle/HOL.

- Axiomatization can introduce an inconsistency.
- Example: After declaring the existence of a new type *SET* in Isabelle, it is possible to add a new axiom:

  axiomatization
  $\quad Member :: SET \Rightarrow SET \Rightarrow bool$
  where
  $\quad comprehension : \exists y. \forall x.\ Member\ x\ y \longleftrightarrow P\ x$

  which enables a "proof" of the paradoxical lemma:

  lemma $member\_iff\_not\_member : \exists y.\ Member\ y\ y \longleftrightarrow \neg Member\ y\ y$

  from which *False* can be derived.

Axiomatic Extensions Considered Harmful

As we saw already, **definitional** extension is favoured over **axiomatic** extension in Isabelle/HOL.

- Axiomatization can introduce an inconsistency.
- Example: After declaring the existence of a new type *SET* in Isabelle, it is possible to add a new axiom:

  axiomatization
  $\quad Member :: SET \Rightarrow SET \Rightarrow bool$
  where
  $\quad comprehension : \exists y. \forall x. \; Member\ x\ y \longleftrightarrow P\ x$

  which enables a "proof" of the paradoxical lemma:

  lemma $member\_iff\_not\_member : \exists y. \; Member\ y\ y \longleftrightarrow \neg Member\ y\ y$

  from which *False* can be derived.
- Yet, axiomatic reasoning is part of mathematics. We want to be able to carry it out safely in Isabelle.

Local axiomatic reasoning in Isabelle/HOL

Fortunately, we can reason from axioms *locally* in a sound way. For example, to prove results about groups, rings or vector spaces.

Local axiomatic reasoning in Isabelle/HOL

Fortunately, we can reason from axioms *locally* in a sound way. For example, to prove results about groups, rings or vector spaces.

We later *instantiate* the axioms with actual groups, rings, vector spaces.

Local axiomatic reasoning in Isabelle/HOL

Fortunately, we can reason from axioms *locally* in a sound way. For example, to prove results about groups, rings or vector spaces.

We later *instantiate* the axioms with actual groups, rings, vector spaces.

Isabelle provides a facility for doing this called **locales**.

```
locale group =
  fixes mult :: 'a ⇒ 'a ⇒ 'a   and   unit :: 'a
  assumes left_unit      : mult unit x = x
      and associativity : mult x (mult y z) = mult (mult x y) z
      and left_inverse  : ∃y. mult y x = unit
```

Local axiomatic reasoning in Isabelle/HOL

Fortunately, we can reason from axioms *locally* in a sound way. For example, to prove results about groups, rings or vector spaces.

We later *instantiate* the axioms with actual groups, rings, vector spaces.

Isabelle provides a facility for doing this called **locales**.

$$
\begin{array}{l}
\texttt{locale}\ \mathit{group} = \\
\quad \texttt{fixes}\ \mathrm{mult}\ ::\ 'a \Rightarrow 'a \Rightarrow 'a \quad \texttt{and} \quad \mathrm{unit} :: 'a \\
\quad \texttt{assumes}\ \mathit{left\_unit} \quad : \mathrm{mult\ unit}\ x = x \\
\qquad \texttt{and}\ \mathit{associativity} : \mathrm{mult}\ x\ (\mathrm{mult}\ y\ z) = \mathrm{mult}\ (\mathrm{mult}\ x\ y)\ z \\
\qquad \texttt{and}\ \mathit{left\_inverse} \ : \exists y.\ \mathrm{mult}\ y\ x = \mathrm{unit}
\end{array}
$$

- In the above, *mult* and *unit* are just arbitrary names.

Local axiomatic reasoning in Isabelle/HOL

Fortunately, we can reason from axioms *locally* in a sound way. For example, to prove results about groups, rings or vector spaces.

We later *instantiate* the axioms with actual groups, rings, vector spaces.

Isabelle provides a facility for doing this called **locales**.

$$
\begin{array}{l}
\texttt{locale}\ \mathit{group} = \\
\quad \texttt{fixes}\ \mathrm{mult}\ ::\ 'a \Rightarrow 'a \Rightarrow 'a \quad \texttt{and} \quad \mathrm{unit} :: 'a \\
\quad \texttt{assumes}\ \mathit{left\_unit} \quad : \mathrm{mult\ unit}\ x = x \\
\qquad \texttt{and}\ \mathit{associativity} : \mathrm{mult}\ x\ (\mathrm{mult}\ y\ z) = \mathrm{mult}\ (\mathrm{mult}\ x\ y)\ z \\
\qquad \texttt{and}\ \mathit{left\_inverse} \ : \exists y.\ \mathrm{mult}\ y\ x = \mathrm{unit}
\end{array}
$$

- In the above, *mult* and *unit* are just arbitrary names.
- For example, the integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ form a group under the operation of addition i.e. we can instantiate mult to + and unit to 0. More on instantiation later.

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.
  - Context as a formula:

$$\bigwedge \overbrace{x_1 \ldots x_n}^{\textit{parameters}} . \llbracket \overbrace{A_1; \ldots A_m}^{\textit{assumptions}} \rrbracket \Longrightarrow \overbrace{C}^{\textit{theorem}}$$

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.
  - Context as a formula:

$$\bigwedge \overbrace{x_1 \ldots x_n}^{parameters} . [\![\overbrace{A_1; \ldots A_m}^{assumptions}]\!] \Longrightarrow \overbrace{C}^{theorem}$$

- Locales usually have

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.
  - Context as a formula:

$$\bigwedge \overbrace{x_1 \ldots x_n}^{parameters} . [\![\overbrace{A_1; \ldots A_m}^{assumptions}]\!] \Longrightarrow \overbrace{C}^{theorem}$$

- Locales usually have
  - parameters, declared using `fixes`

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.
  - Context as a formula:

$$\bigwedge \overbrace{x_1 \ldots x_n}^{parameters} . \llbracket \overbrace{A_1; \ldots A_m}^{assumptions} \rrbracket \Longrightarrow \overbrace{C}^{theorem}$$

- Locales usually have
  - parameters, declared using `fixes`
  - assumptions, declared using `assumes`

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.
  - Context as a formula:

$$\bigwedge \overbrace{x_1 \ldots x_n}^{\textit{parameters}} . \llbracket \overbrace{A_1; \ldots A_m}^{\textit{assumptions}} \rrbracket \Longrightarrow \overbrace{C}^{\textit{theorem}}$$

- Locales usually have
  - parameters, declared using `fixes`
  - assumptions, declared using `assumes`
- Inside a locale, definitions can be made and theorems proven based on the parameters and assumptions.

Isabelle Locales

- Named, encapsulated contexts, highly suitable for formalising abstract mathematics.
  - Context as a formula:

$$\bigwedge \overbrace{x_1 \ldots x_n}^{\textit{parameters}} . [\![\overbrace{A_1; \ldots A_m}^{\textit{assumptions}}]\!] \Longrightarrow \overbrace{C}^{\textit{theorem}}$$

- Locales usually have
  - parameters, declared using `fixes`
  - assumptions, declared using `assumes`
- Inside a locale, definitions can be made and theorems proven based on the parameters and assumptions.
- A locale can import/extend other locales.

Locale Example: Finite Graphs

$$
\begin{array}{l}
\texttt{locale}\ \textit{finitegraph} = \\
\quad \texttt{fixes}\ \text{edges} :: ({}'a \times {}'a)\ \textit{set} \quad \texttt{and} \quad \text{vertices} :: {}'a\ \textit{set} \\
\quad \texttt{assumes}\ \textit{finite\_vertex\_set} : \text{finite vertices} \\
\qquad \texttt{and}\ \textit{is\_graph} \qquad : (\text{u}, \text{v}) \in \text{edges} \Longrightarrow \text{u} \in \text{vertices} \wedge \text{v} \in \text{vertices}
\end{array}
$$

$$
\begin{array}{l}
\texttt{begin} \\
\quad \texttt{inductive}\ \textit{walk} :: {}'a\ \textit{list} \Rightarrow \textit{bool} \quad \texttt{where} \\
\quad \textit{Nil} \qquad\qquad : \text{walk}\ [] \\
\quad |\textit{Singleton} \quad : \text{v} \in \text{vertices} \Longrightarrow \text{walk}\ [\text{v}] \\
\quad |\textit{Cons} \qquad : [\![(\text{v}, \text{w}) \in \text{edges}; \text{walk}(\text{w}\#\text{vs})]\!] \Longrightarrow \text{walk}\ (\text{v}\#\text{w}\#\text{vs}) \\
\quad \texttt{lemma}\ \textit{walk\_edge} : (\text{v}, \text{w}) \in \text{edges} \Longrightarrow \text{walk}\ [\text{v}, \text{w}] \\
\quad \ldots \\
\texttt{end}
\end{array}
$$

- # is the list cons operator in Isabelle.

Locale Example: Finite Graphs

```
locale finitegraph =
  fixes edges :: ('a × 'a) set   and   vertices :: 'a set
  assumes finite_vertex_set : finite vertices
     and is_graph           : (u, v) ∈ edges ⟹ u ∈ vertices ∧ v ∈ vertices
begin

  inductive walk :: 'a list ⇒ bool   where
  Nil          :  walk []
  |Singleton   :  v ∈ vertices  ⟹  walk [v]
  |Cons        :  ⟦(v, w) ∈ edges; walk(w#vs)⟧  ⟹  walk (v#w#vs)

 lemma walk_edge : (v, w) ∈ edges  ⟹  walk [v, w]
 ...
end
```

- # is the list cons operator in Isabelle.
- The definition of this locale can be inspected by typing **thm** *finitegraph_def* in Isabelle:

$$\text{finitegraph } ?edges\ ?vertices \equiv$$
$$\text{finite } ?vertices \wedge$$
$$(\forall uv.(u, v) \in ?edges \longrightarrow u \in ?vertices \wedge v \in ?vertices)$$

Adding Theorems to a Locale

Aside from proving a lemma within the locale definition, e.g. *walk_edge* on the previous slide, we can also state lemmas that are `"in"` some locale:

```
lemma (in group) associativity_bw :
    "mult (mult x y) z = mult x (mult y z)"
apply (subst associativity)
apply (rule refl)
done
```

Adding Theorems to a Locale

Aside from proving a lemma within the locale definition, e.g. *walk_edge* on the previous slide, we can also state lemmas that are "in" some locale:

```
lemma (in group) associativity_bw :
    "mult (mult x y) z = mult x (mult y z)"
apply (subst associativity)
apply (rule refl)
done
```

Alternatively, we can enter a locale at the theory level using the **context** keyword and formalize new definitions and theorems:

```
context group
begin
    lemma associativity_bw :
        "mult (mult x y) z = mult x (mult y z)"
    apply (subst associativity)
    apply (rule refl)
    done
end
```

Locale Extension

- New locales can extend existing ones by adding more parameter, assumptions and definitions. This is also known as an *import*.

Locale Extension

- New locales can extend existing ones by adding more parameter, assumptions and definitions. This is also known as an *import*.
- The context of the imported locale i.e. all its assumptions, theorems etc. are available in the extended locale.

Locale Extension

- New locales can extend existing ones by adding more parameter, assumptions and definitions. This is also known as an *import*.
- The context of the imported locale i.e. all its assumptions, theorems etc. are available in the extended locale.

$$\texttt{locale}\ \textit{weighted\_finitegraph} = \textit{finitegraph}\ +$$
$$\quad \texttt{fixes}\ \text{weight}\ ::\ ('a \times 'a) \Rightarrow \textit{nat}$$
$$\quad \texttt{assumes}\ \textit{edges\_weighted} : \forall \text{e} \in \text{edges}. \exists \text{w}.\ \text{weight}\ \text{e} = \text{w}$$

Locale Extension

- New locales can extend existing ones by adding more parameter, assumptions and definitions. This is also known as an *import*.
- The context of the imported locale i.e. all its assumptions, theorems etc. are available in the extended locale.

$$\texttt{locale}\ \textit{weighted\_finitegraph} = \textit{finitegraph}\ +$$
$$\texttt{fixes}\ \text{weight}\ :: (\prime a \times \prime a) \Rightarrow \textit{nat}$$
$$\texttt{assumes}\ \textit{edges\_weighted} : \forall \text{e} \in \text{edges}.\exists \text{w}.\ \text{weight e} = \text{w}$$

Viewed in terms of the imported *finitegraph* locale (and the weighted edges axiom), we have:

$$\text{weighted\_finitegraph}\ ?\textit{edges}\ ?\textit{vertices}\ ?\textit{weight} \equiv$$
$$\text{finitegraph}\ ?\text{edges}\ ?\text{vertices} \wedge (\forall e \in ?\textit{edges}.\ \exists w.\ ?\textit{weight}\ e = w)$$

Instantiating Locales

- *Concrete* examples may be proven to be instances of a locale.

Instantiating Locales

- *Concrete* examples may be proven to be instances of a locale.
- interpretation `interpretation_name : locale_name args` generates the proof obligation that the locale predicate holds of the `args`.

Instantiating Locales

- *Concrete* examples may be proven to be instances of a locale.
- interpretation `interpretation_name : locale_name args` generates the proof obligation that the locale predicate holds of the `args`.
- Example: A graph with one vertex and single edge from that vertex to itself is a concrete instance of the locale *finite_graph*.

Instantiating Locales

- *Concrete* examples may be proven to be instances of a locale.
- interpretation `interpretation_name : locale_name args` generates the proof obligation that the locale predicate holds of the `args`.
- Example: A graph with one vertex and single edge from that vertex to itself is a concrete instance of the locale *finite_graph*.

```
interpretation singleton_finitegraph : finitegraph "{(1, 1)}" "{1}"
proof
  show "finite {1}" by simp
  next fix u v
  assume "(u, v) ∈ {(1, 1)}" then show "u ∈ {1} ∧ v ∈ {1}" by blast
qed
```

Instantiating Locales

- *Concrete* examples may be proven to be instances of a locale.
- interpretation `interpretation_name : locale_name args` generates the proof obligation that the locale predicate holds of the `args`.
- Example: A graph with one vertex and single edge from that vertex to itself is a concrete instance of the locale *finite_graph*.

```
interpretation singleton_finitegraph : finitegraph "{(1, 1)}" "{1}"
proof
  show "finite {1}" by simp
  next fix u v
  assume "(u, v) ∈ {(1, 1)}" then show "u ∈ {1} ∧ v ∈ {1}" by blast
qed
```

- We can prove that *singleton_finitegraph* is an instance of a finite weighted graph locale by providing a weight function as an additional argument:

Instantiating Locales

- *Concrete* examples may be proven to be instances of a locale.
- interpretation `interpretation_name : locale_name args` generates the proof obligation that the locale predicate holds of the `args`.
- Example: A graph with one vertex and single edge from that vertex to itself is a concrete instance of the locale *finite_graph*.

```
interpretation singleton_finitegraph : finitegraph "{(1, 1)}" "{1}"
proof
  show "finite {1}" by simp
  next fix u v
  assume "(u, v) ∈ {(1, 1)}" then show "u ∈ {1} ∧ v ∈ {1}" by blast
qed
```

- We can prove that *singleton_finitegraph* is an instance of a finite weighted graph locale by providing a weight function as an additional argument:

```
interpretation
 singleton_finitegraph : weighted_finitegraph "{(1, 1)}" "{1}" "λ(u, v). 1"
 by (unfold_locales) simp
```

Summary

- Axiomatization at the Isabelle theory level (i.e. as an extension of Isabelle/HOL) is not favoured as it can be unsound (see the additional exercise on the AR web page).
- Locales provide a sound way of reasoning locally about axiomatic theories.
- This was an introduction to locale declarations, extensions and interpretations.
  - There are many other features involving representation and reasoning using locales in Isabelle.
  - Reading: Tutorial on Locales and Locale Interpretation (on the AR Lecture Schedule page in Learn).