Automated Reasoning

Lecture 8: Representation II
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
  Member :: SET ⇒ SET ⇒ bool
where
  comprehension : ∃y.∀x. Member x y ⟷ 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 \texttt{SET} in Isabelle, it is possible to add a new axiom:

  
  \[
  \text{axiomatization} \\
  \quad \text{Member} :: \text{SET} \Rightarrow \text{SET} \Rightarrow \text{bool} \\
  \quad \text{where} \\
  \quad \text{comprehension} : \exists y. \forall x. \text{Member} x y \iff P x
  \]

  which enables a "proof" of the paradoxical lemma:

  \[
  \text{lemma member iff not member} : \exists y. \text{Member} y y \iff \neg \text{Member} y y
  \]

  from which \textit{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 $\text{SET}$ in Isabelle, it is possible to add a new axiom:

  ``` Isabelle
  axiomatization
  Member :: $\text{SET} \Rightarrow \text{SET} \Rightarrow \text{bool}$
  where
  comprehension : $\exists y. \forall x. \text{Member } x \ y \leftrightarrow P \ x$
  ```

  which enables a "proof" of the paradoxical lemma:

  ``` Isabelle
  lemma member_iff_not_member : $\exists y. \text{Member } y \ y \leftrightarrow \neg \text{Member } y \ y$
  ```

  from which $\text{False}$ can be derived.

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

```plaintext
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
```
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:
    \[
    \bigwedge x_1 \ldots x_n \left[ A_1; \ldots A_m \right] \implies C
    \]
Isabelle Locales

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

\[
\bigwedge_{x_1 \ldots x_n} [A_1; \ldots A_m] \Rightarrow C
\]

- Locales usually have
Isabelle Locales

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

\[
\bigwedge \, \text{parameters} \quad \underbrace{\text{assumptions}} \quad \text{theorem} \\
\quad x_1 \ldots x_n . \left[ A_1; \ldots A_m \right] \implies C
\]

- Locales usually have
  - parameters, declared using \textit{fixes}
Isabelle Locales

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

\[
\forall x_1 \ldots x_n \left[ A_1 ; \ldots ; A_m \right] \Rightarrow C
\]

- 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:
    \[
    \bigwedge_{x_1 \ldots x_n} [A_1; \ldots; A_m] \Rightarrow C
    \]

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

Context as a formula:

\[ \bigwedge_{x_1 \ldots x_n} [A_1; \ldots A_m] \implies C \]

Locales usually have

- parameters, declared using \texttt{fixes}
- assumptions, declared using \texttt{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

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.
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:

  finitegraph ?edges ?vertices ≡ 
  finite ?vertices ∧ 
  (∀uv.(u, v) ∈ ?edges → u ∈ ?vertices ∧ v ∈ ?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:

```plaintext
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. \textit{walk_edge} on the previous slide, we can also state lemmas that are "in" some locale:

```latex
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 \texttt{context} keyword and formalize new definitions and theorems:

```latex
context group
begin

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

end
```
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.
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.

```plaintext
locale weighted_finitegraph = finitegraph +
  fixes weight :: ('a × 'a) ⇒ nat
  assumes edges_weighted : ∀ e ∈ edges. ∃ w. weight e = 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.

```plaintext
locale weighted_finitegraph = finitegraph +
  fixes weight :: ('a × 'a) ⇒ nat
  assumes edges_weighted : ∀e ∈ edges. ∃w. weight e = w
```

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

```plaintext
weighted_finitegraph ?edges ?vertices ?weight ≡
finitegraph ?edges ?vertices ∧ (∀e ∈ ?edges. ∃w. ?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`.

Example: A graph with one vertex and single edge from that vertex to itself is a concrete instance of the locale `finite_graph`.

```plaintext
interpretation singleton_finitegraph : finitegraph 
proof
show "finite f1g" by simp 
next
fix u v 
assume "(u; v) 2 f1g" then show "u 2 f1g^ v 2 f1g" 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:

```plaintext
interpretation singleton_finitegraph : weighted_finitegraph 
proof
by (unfold_locales simp)
```
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 \textit{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`.

```isar
interpretation singleton_finitegraph : finitegraph "\{(1, 1)\}" "\{1\}"
proof
  show "finite \{1\}" by simp
next fix u v
  assume "(u, v) \in \{(1, 1)\}" then show "u \in \{1\} \land v \in \{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`.

```plaintext
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`.

```plaintext
interpretation singleton_finitegraph : finitegraph "\{(1, 1)\}" "\{1\}"
proof
  show "finite \{1\}" by simp
next fix u v
  assume "(u, v) \in \{(1, 1)\}" then show "u \in \{1\} \land v \in \{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:

```plaintext
interpretation
  singleton_finitegraph : weighted_finitegraph "\{(1, 1)\}" "\{1\}" "\lambda(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 to Locales and Locale Interpretation (on the AR web page).