Automated Reasoning

Lecture 7: Locales in Isabelle/HOL

Jacques Fleuriot jdf@inf.ed.ac.uk

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

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

Axiomatization can introduce an inconsistency.

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 \Rightarrow SET \Rightarrow bool

where

comprehension : \exists y. \forall x. Member x y \longleftrightarrow P x
```

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 \Rightarrow SET \Rightarrow bool

where

comprehension : \exists y. \forall x. Member x y \longleftrightarrow P x
```

which enables a "proof" of the paradoxical lemma:

 $\texttt{lemma member_iff_not_member}: \exists y. \ \textit{Member } y \ y \longleftrightarrow \neg \textit{Member } y \ y$

from which False can be derived.

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 \Rightarrow SET \Rightarrow bool

where

comprehension : \exists y. \forall x. Member x y \longleftrightarrow P x
```

which enables a "proof" of the paradoxical lemma:

 $\texttt{lemma member_iff_not_member: \exists y. Member y y \longleftrightarrow \neg Member y y}$

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.

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

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.

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 \Rightarrow 'a \Rightarrow '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 : $\exists y$. mult y x = unit

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 \Rightarrow 'a \Rightarrow '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 : $\exists y$. mult y x = unit

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

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**.

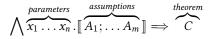
locale group =
fixes mult :: 'a \Rightarrow 'a \Rightarrow 'a and unit :: 'a
assumes left_unit : mult unit x = x
and associativity : mult x (mult y z) = mult (mult x y) z
and left_inverse : $\exists y$. mult y x = unit

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

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

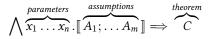
 Named, encapsulated contexts, highly suitable for formalising abstract mathematics.

Context as a formula:



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

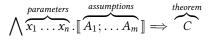
Context as a formula:



Locales usually have

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

Context as a formula:

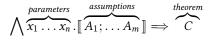


Locales usually have

parameters, declared using fixes

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

Context as a formula:

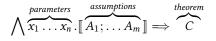


Locales usually have

- parameters, declared using fixes
- assumptions, declared using assumes

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

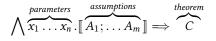
Context as a formula:



- 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:



- 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

```
locale finitegraph =
  fixes edges :: (a \times a) set and vertices :: a set
  assumes finite_vertex_set : finite vertices
       and is graph : (u, v) \in edges \implies u \in vertices \land v \in vertices
begin
    inductive walk :: 'a list \Rightarrow bool where
    Nil : walk []
    |Singleton : v \in vertices \implies walk[v]
    Cons
                    : [(v, w) \in edges; walk(w \# vs)] \implies walk(v \# w \# vs)
 lemma walk edge: (v, w) \in edges \implies walk [v, w]
 ...
end
```

is the list cons operator in Isabelle.

Locale Example: Finite Graphs

```
locale finitegraph =
  fixes edges :: (a \times a) set and vertices :: a set
  assumes finite_vertex_set : finite vertices
       and is graph : (u, v) \in edges \implies u \in vertices \land v \in vertices
begin
    inductive walk :: 'a list \Rightarrow bool where
    Nil : walk []
    |\textit{Singleton} \quad : \ v \in vertices \implies walk[v]
    Cons
                     : [(v, w) \in edges; walk(w \# vs)] \implies walk(v \# w \# vs)
 lemma walk edge: (v, w) \in edges \implies 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 \equiv
finite ?vertices \land
(\forall uv.(u, v) \in ?edges \longrightarrow u \in ?vertices \land 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
```

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

- 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.

locale weighted_finitegraph = finitegraph + fixes weight :: $('a \times 'a) \Rightarrow nat$ assumes edges_weighted : $\forall e \in edges. \exists w. weight e = w$

- 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 weighted_finitegraph = finitegraph + fixes weight :: $('a \times 'a) \Rightarrow nat$ assumes edges_weighted : $\forall e \in edges. \exists w. weight e = w$

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

weighted_finitegraph ?edges ?vertices ?weight \equiv finitegraph ?edges ?vertices $\land (\forall e \in ?edges. \exists w. ?weight e = w)$

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

- *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.

- *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*.

qed

- *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

- *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:

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

```
\label{eq:interpretation} 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 on Locales and Locale Interpretation (on the AR Lecture Schedule page in Learn).