Representing Isabelle in LF

Florian Rabe

Jacobs University Bremen

Slogans

- Type classes are wrong:
  Type classes should be theories, instances should be morphisms.
- The Isabelle module system is too complicated:
  You do not need theories, locales, *and* type classes.
- The LF module system is good:
  - LF = judgments as types, proofs as terms
  - LF module system =
    inference systems as signatures, relations as morphisms
  - simple, elegant, expressive

Background

- Long term goal:
  - comprehensive framework to represent, integrate, translate, reason about logics
  - apply to all commonly used logics, generate large content base
    digital library of logics
  - cover model and proof theory
  - provide tool support: validation, browsing, editing, storage, ...
- State:
  - successful progress based on modular Twelf
    twelf-mod branch of Twelf
  - fast-growing library https://trac.omdoc.org/LATIN/
  - besides logics: set theory, λ-cube, Mizar, **Isabelle**/HOL, ...

Overview

- Designed representation of Isabelle in LF

  an outsider's account of Isabelle

  - includes type classes, locales, theories, excludes Isar
  - yields concise formal definition of Isabelle
  - complements Isabelle documentation
- Next steps require inside support
  - better statement and proof of adequacy
  - implementation

Isabelle

theory	::=	**theory** T **imports** T^* **begin** *thycont* **end**
thycont	::=	(*locale* | *sublocale* | *interpretation* |
	|	*class* | *instantiation* | *thysymbol*)*
locale	::=	**locale** $L = (i :$ *instance*$)^*$ for *locsymbol** + *locsymbol**
sublocale	::=	**sublocale** $L <$ *instance proof**
interpretation	::=	**interpretation** *instance proof**
instance	::=	L **where** *namedinst**
class	::=	**class** $C = C^* +$ *locsymbol**
instantiation	::=	**instantiation** *type* :: $(C^*)C$ **begin** *locsymbol** *proof** **end**
thysymbol	::=	**consts** *con* | **defs** *def* | **axioms** *ax* | **lemma** *lem*
	|	**typedecl** *typedecl* | **types** *types*
locysymbol	::=	**fixes** *con* | **defines** *def* | **assumes** *ax* | **lemma** *lem*
con	::=	c :: *type*
def	::=	$a : c\ x^* \equiv$ *term*
ax	::=	a : *form*
lem	::=	a : *form proof*
typedecl	::=	$(\alpha^*)t$ *name*
types	::=	$(\alpha^*)t =$ *type*
namedinst	::=	$c =$ *term*
type	::=	$\alpha :: C$ | (*type**) t | *type* $\Rightarrow$ *type* | $\mathtt{prop}$
term	::=	x | c | *term term* | $\lambda(x ::$ *type*$)^*$.*term*
form	::=	*form* $\Longrightarrow$ *form* | $\bigwedge(x ::$ *type*$)^*$.*form* | *term* $\equiv$ *term*
proof	::=	a primitive Pure inference as described in the manual

Representing the Primitives

$$
\begin{array}{llll}
\texttt{sig}\ \mathit{Pure} = \{ & & & \\
\quad \mathit{tp} & : & \texttt{type}. & \\
\quad \Rightarrow & : & \mathit{tp} \longrightarrow \mathit{tp} \longrightarrow \mathit{tp}. & \texttt{infix}\ \mathsf{right}\ 0 \Rightarrow. \\
\quad \mathit{tm} & : & \mathit{tp} \longrightarrow \texttt{type}. & \texttt{prefix}\ 0\ \mathit{tm}. \\
\quad \lambda & : & (\mathit{tm}\ A \longrightarrow \mathit{tm}\ B) \longrightarrow \mathit{tm}\ (A \Rightarrow B). & \\
\quad @ & : & \mathit{tm}\ (A \Rightarrow B) \longrightarrow \mathit{tm}\ A \longrightarrow \mathit{tm}\ B. & \texttt{infix}\ \mathsf{left}\ 1000\ @. \\
 & & & \\
\quad \texttt{prop} & : & \mathit{tp}. & \\
\quad \bigwedge & : & (\mathit{tm}\ A \longrightarrow \mathit{tm}\ \texttt{prop}) \longrightarrow \mathit{tm}\ \texttt{prop}. & \\
\quad \Longrightarrow & : & \mathit{tm}\ \texttt{prop} \longrightarrow \mathit{tm}\ \texttt{prop} \longrightarrow \mathit{tm}\ \texttt{prop}. & \texttt{infix}\ \mathsf{right}\ 1 \Longrightarrow. \\
\quad \equiv & : & \mathit{tm}\ A \longrightarrow \mathit{tm}\ A \longrightarrow \mathit{tm}\ \texttt{prop}. & \texttt{infix}\ \mathsf{none}\ 2 \equiv. \\
 & & & \\
\quad \vdash & : & \mathit{tm}\ \texttt{prop} \longrightarrow \texttt{type}. & \texttt{prefix}\ 0 \vdash. \\
\quad \bigwedge \mathsf{I} & : & (x : \mathit{tm}\ A \vdash (B\ x)) \longrightarrow \vdash \bigwedge([x]B\ x). & \\
\quad \bigwedge \mathsf{E} & : & \vdash \bigwedge([x]B\ x) \longrightarrow \{x : \mathit{tm}\ A\} \vdash (B\ x). & \\
\quad \Longrightarrow \mathsf{I} & : & (\vdash A \longrightarrow \vdash B) \longrightarrow \vdash A \Longrightarrow B. & \\
\quad \Longrightarrow \mathsf{E} & : & \vdash A \Longrightarrow B \longrightarrow \vdash A \longrightarrow \vdash B. & \\
\quad \mathsf{refl} & : & \vdash X \equiv X. & \\
\quad \mathsf{subs} & : & \{F : \mathit{tm}\ A \longrightarrow \mathit{tm}\ B\} \vdash X \equiv Y \longrightarrow \vdash F\ X \equiv F\ Y. & \\
\quad \mathsf{exten} & : & \{x : \mathit{tm}\ A\} \vdash (F\ x) \equiv (G\ x) \longrightarrow \vdash \lambda F \equiv \lambda G. & \\
\quad \mathsf{beta} & : & \vdash (\lambda[x : \mathit{tm}\ A]F\ x)\ @\ X \equiv F\ X. & \\
\quad \mathsf{eta} & : & \vdash \lambda\ ([x : \mathit{tm}\ A]F\ @\ x) \equiv F. & \\
\quad \texttt{sig}\ \mathit{Type} & = & \{\mathit{this} : \mathit{tp}.\}. & \\
\}. & & &
\end{array}
$$

Representing Simple Expressions

Expression	Isabelle	LF
type operator	$(\alpha_1, \ldots, \alpha_n)\ t$	$t : tp \to \ldots \to tp \to tp$
type variable	α	$\alpha : tp$
constant	$c :: \tau$	$c : tm\ \ulcorner\tau\urcorner$
variable	$x :: \tau$	$x : tm\ \ulcorner\tau\urcorner$
assumption/axiom	$a : \varphi$	$a : \vdash \ulcorner\varphi\urcorner$
lemma/theorem	$a : \varphi\ P$	$a : \vdash \ulcorner\varphi\urcorner = \ulcorner P\urcorner$

Polymorphism: τ, φ, P may contain type variables $\alpha_1, \ldots, \alpha_n$
Represented as LF binding, e.g.,

$$a : \{\alpha_1 : tp\} \ldots \{\alpha_n : tp\}\ \vdash \ulcorner\varphi\urcorner = [\alpha_1 : tp] \ldots [\alpha_n : tp] \ulcorner P\urcorner$$

Isabelle

theory	$::=$	**theory** T **imports** T^* **begin** *thycont* **end**
thycont	$::=$	$($*locale* $\mid$ *sublocale* $\mid$ *interpretation* $\mid$
	$\mid$	*class* $\mid$ *instantiation* $\mid$ *thysymbol*$)^*$
locale	$::=$	**locale** $L = (i :$ *instance*$)^*$ for *locsymbol** $+$ *locsymbol**
sublocale	$::=$	**sublocale** $L <$ *instance proof**
interpretation	$::=$	**interpretation** *instance proof**
instance	$::=$	L **where** *namedinst**
class	$::=$	**class** $C = C^* +$ *locsymbol**
instantiation	$::=$	**instantiation** *type* $::$ $(C^*)C$ **begin** *locsymbol** *proof** **end**
thysymbol	$::=$	**consts** *con* $\mid$ **axioms** *ax* $\mid$ **lemma** *lem* $\mid$ **typedecl** *typedecl*
locysymbol	$::=$	**fixes** *con* $\mid$ **assumes** *ax* $\mid$ **lemma** *lem*
con	$::=$	$c ::$ *type*
ax	$::=$	$a :$ *form*
lem	$::=$	$a :$ *form proof*
typedecl	$::=$	$(\alpha^*)t$ *name*
namedinst	$::=$	$c =$ *term*

Isabelle

theory	::=	**theory** T **imports** T^* **begin** *thycont* **end**
thycont	::=	(*locale* | *sublocale* | *interpretation* |
	|	*class* | *instantiation* | *thysymbol*)*
locale	::=	**locale** $L = (i :$ *instance*$)^*$ for *locsymbol** + *locsymbol**
sublocale	::=	**sublocale** $L <$ *instance proof**
interpretation	::=	**interpretation** *instance proof**
instance	::=	L **where** *namedinst**
class	::=	**class** $C = C^* +$ *locsymbol**
instantiation	::=	**instantiation** *type* :: $(C^*)C$ **begin** *locsymbol** *proof** **end**
thysymbol	::=	**consts** *con* | **axioms** *ax* | **lemma** *lem* | **typedecl** *typedecl*
locysymbol	::=	**fixes** *con* | **assumes** *ax* | **lemma** *lem*
con	::=	c :: *type*
ax	::=	a : *form*
lem	::=	a : *form proof*
typedecl	::=	$(\alpha^*)t$ *name*
namedinst	::=	c = *term*

3 scoping constructs

Isabelle

theory	::=	**theory** *T* **imports** *T** **begin** *thycont* **end**
thycont	::=	(*locale* | *sublocale* | *interpretation* |
	|	*class* | *instantiation* | *thysymbol*)*
locale	::=	**locale** *L* = (*i* : *instance*)* for *locsymbol** + *locsymbol**
sublocale	::=	**sublocale** *L* < *instance proof**
interpretation	::=	**interpretation** *instance proof**
instance	::=	*L* **where** *namedinst**
class	::=	**class** *C* = *C** + *locsymbol**
instantiation	::=	**instantiation** *type* :: (*C**)*C* **begin** *locsymbol** *proof** **end**
thysymbol	::=	**consts** *con* | **axioms** *ax* | **lemma** *lem* | **typedecl** *typedecl*
locysymbol	::=	**fixes** *con* | **assumes** *ax* | **lemma** *lem*
con	::=	*c* :: *type*
ax	::=	*a* : *form*
lem	::=	*a* : *form proof*
typedecl	::=	(α^*)*t name*
namedinst	::=	*c* = *term*

3 scoping constructs with one import declaration each

Isabelle

theory	::=	**theory** *T* **imports** *T** **begin** *thycont* **end**
thycont	::=	(*locale* | *sublocale* | *interpretation* |
	|	*class* | *instantiation* | *thysymbol*)*
locale	::=	**locale** *L* = (*i* : *instance*)* for *locsymbol** + *locsymbol**
sublocale	::=	**sublocale** *L* < *instance proof**
interpretation	::=	**interpretation** *instance proof**
instance	::=	*L* **where** *namedinst**
class	::=	**class** *C* = *C** + *locsymbol**
instantiation	::=	**instantiation** *type* :: (*C**)*C* **begin** *locsymbol** *proof** **end**
thysymbol	::=	**consts** *con* | **axioms** *ax* | **lemma** *lem* | **typedecl** *typedecl*
locysymbol	::=	**fixes** *con* | **assumes** *ax* | **lemma** *lem*
con	::=	*c* :: *type*
ax	::=	*a* : *form*
lem	::=	*a* : *form proof*
typedecl	::=	(α^*)*t name*
namedinst	::=	*c* = *term*

3 scoping constructs with one import declaration each

3 constructs to relate scopes

LF

$$
\begin{array}{llcl}
\text{Signatures} & \Sigma & ::= & \cdot \mid \Sigma,\ \texttt{sig}\ T\ =\ \{\Sigma\} \mid \Sigma,\ \texttt{view}\ v\ :S\ \to T\ = \mu \mid \Sigma,\ \texttt{include}\ S \\
& & & \mid \Sigma,\ \texttt{struct}\ s\ :S\ =\{\sigma\} \mid \Sigma,\ c:A \mid \Sigma,\ a:K \\
\text{Morphisms} & \sigma & ::= & \cdot \mid \sigma,\ \texttt{struct}\ s := \mu \quad \mid \sigma,\ c := t \quad \mid \sigma,\ a := A \\
& \mu & ::= & \{\sigma\} \mid v \mid incl \mid s \mid id \mid \mu\ \mu \\
\text{Kinds} & K & ::= & \texttt{type} \mid \{x:A\}\ K \\
\text{Type families} & A & ::= & a \mid \quad [x:A]\ A \mid A\ t \mid \{x:A\}\ A \\
\text{Terms} & t & ::= & c \mid x \mid \quad [x:A]\ t \mid t\ t
\end{array}
$$

- Signatures scope declarations: signatures, morphisms, constants, type families
- Morphisms relate signatures:
  - `view`: explicit morphism
  - `include`: inclusion into current signature
  - `struct`: named import into current signature

Morphisms

- A morphisms relates two signatures
- Morphism from S to T
  - maps S constants to T-terms
  - maps S type family symbols to T-type families
  - extends homomorphically to all S-expressions
  - preserves typing, kinding, definitional equality
- `view` $v : S \to T = \{\sigma\}$: maps given explicitly by σ
- `include` S: inclusion from S into current signature
- `struct` $s : S = \{\sigma\}$: named import from S into current signature, maps c to $s.c$

Representing Theories

Isabelle:

theory T **imports** $T_1, \ldots, T_n$ **begin** Σ **end**

LF representation:

$\mathtt{sig}\ T = \{\mathtt{include}\ \mathit{Pure}, \mathtt{include}\ T_1, \ldots, \mathtt{include}\ T_n, \ulcorner\Sigma\urcorner\}.$

Representing Modular Declarations

Scopes as signatures, relations as morphisms

Isabelle	LF
theory	signature
locale	signature
type class	signature
theory import	morphism (inclusion)
locale import from L	morphism (structure from S)
type class import from C	morphism (structure from C)
sublocale L' of L	morphism (view from L to L')
interpretation of L in T	morphism (view from L to T)
instance of type class C	morphism out of C
type class functor	morphism (view)
type class functor application	morphism composition

Type Classes

Isabelle: types universal for each declaration

$$
\begin{array}{lll}
\textbf{class } \mathit{semlat} = & & \\
\mathit{leq} & :: & \alpha \Rightarrow \alpha \Rightarrow \mathtt{prop} \\
\mathit{inf} & :: & \beta \Rightarrow \beta \Rightarrow \beta \\
\mathit{ax} & : & \bigwedge x : \gamma \bigwedge y : \gamma . \mathit{leq}\ (\mathit{inf}\ x\ y)\ x
\end{array}
$$

LF representation: types existential for all declarations

$$
\begin{array}{lll}
\mathtt{sig}\ \mathit{semlat} & = & \{ \\
\quad \mathit{this} & : & \mathit{tp} \\
\quad \mathit{leq} & : & \mathit{tm}\ (\mathit{this} \Rightarrow \mathit{this} \Rightarrow \mathtt{prop}) \\
\quad \mathit{inf} & : & \mathit{tm}\ (\mathit{this} \Rightarrow \mathit{this} \Rightarrow \mathit{this}) \\
\quad \mathit{ax} & : & \vdash\ (\bigwedge[x : \mathit{this}] \bigwedge[y : \mathit{this}]\ \mathit{leq}\ (\mathit{inf}\ x\ y)\ x) \\
\}
\end{array}
$$

Type Classes

Isabelle: types universal for each declaration

$$\begin{array}{lll} \textbf{class}\ \mathit{semlat} = & & \\ \mathit{leq} & :: & \alpha \Rightarrow \alpha \Rightarrow \mathtt{prop} \\ \mathit{inf} & :: & \beta \Rightarrow \beta \Rightarrow \beta \\ \mathit{ax} & : & \bigwedge x : \gamma \bigwedge y : \gamma . \mathit{leq}\ (\mathit{inf}\ x\ y)\ x \end{array}$$

$$\begin{array}{lll} \multicolumn{3}{l}{\textbf{instantiation}\ \mathit{nat} :: \mathit{semlat}\ \textbf{begin}} \\ \mathit{leq} & = & \leq \\ \mathit{inf} & = & \mathit{min} \\ P & & \\ \textbf{end} & & \end{array}$$

LF representation: types existential for all declarations

$$\begin{array}{lll} \mathtt{sig}\ \mathit{semlat} & = & \{ \\ \quad \mathit{this} & : & \mathit{tp} \\ \quad \mathit{leq} & : & \mathit{tm}\ (\mathit{this} \Rightarrow \mathit{this} \Rightarrow \mathtt{prop}) \\ \quad \mathit{inf} & : & \mathit{tm}\ (\mathit{this} \Rightarrow \mathit{this} \Rightarrow \mathit{this}) \\ \quad \mathit{ax} & : & \vdash\ (\bigwedge [x : \mathit{this}] \bigwedge [y : \mathit{this}]\ \mathit{leq}\ (\mathit{inf}\ x\ y)\ x) \\ \} & & \end{array}$$

$$\begin{array}{lll} \mathtt{view}\ v\ : \mathit{semlat} \to \mathit{Nat} & = & \{ \\ \quad \mathit{this} & := & \mathit{nat} \\ \quad \mathit{leq} & := & \leq \\ \quad \mathit{inf} & := & \mathit{min} \\ \quad \mathit{ax} & := & \ulcorner P \urcorner \\ \} & & \end{array}$$

Type Class Instances as Morphism

- Isabelle intuition:
  - *Type*: class of all types
  - type classes: subclasses of *Type*, predicates on *Type*
- Problem: type classes boring unless associated with operations, say *leq*
- Isabelle solution:
  - *leq* exists at each type
  - each type may define *leq* separately
  - types without definition for *leq* presumably not in the type class
- LF intuition:
  - *Type*: signature $\{\mathit{this} : \mathit{tp}\}$
  - type classes C: signatures extending *Type*
  - type class instances $\tau :: C$ relative to theory/locale L: morphisms

$$\ulcorner \tau :: C \urcorner : C \to L \text{ such that } \ulcorner \tau :: C \urcorner(\mathit{this}) = \ulcorner \tau \urcorner : \mathit{tp}$$

Inheritance between Type Classes

$\textbf{class}\ \mathit{order} =$
$\mathit{leq} \quad :: \quad \alpha \Rightarrow \alpha \Rightarrow \texttt{prop}$

$\textbf{class}\ \mathit{semlat} = \mathit{order} +$
$\mathit{inf} \quad :: \quad \alpha \Rightarrow \alpha \Rightarrow \alpha$

$$
\begin{array}{lll}
\texttt{sig}\ \mathit{order} & = & \{ \\
\quad \mathit{this} & : & \mathit{tp} \\
\quad \mathit{leq} & : & \mathit{tm}\ (\mathit{this} \Rightarrow \mathit{this} \Rightarrow \texttt{prop}) \\
\}
\end{array}
$$

$$
\begin{array}{lll}
\texttt{sig}\ \mathit{semlat} & = & \{ \\
\quad \mathit{this} & : & \mathit{tp} \\
\quad \texttt{struct}\ \mathit{ord} & : \mathit{order} & = \{\mathit{this} := \mathit{this}\} \\
\quad \mathit{inf} & : & \mathit{tm}\ (\mathit{this} \Rightarrow \mathit{this} \Rightarrow \mathit{this}) \\
\}
\end{array}
$$

Inheritance between Type Classes

class *order* =
leq :: $\alpha \Rightarrow \alpha \Rightarrow \texttt{prop}$

class *semlat* = *order* +
inf :: $\alpha \Rightarrow \alpha \Rightarrow \alpha$

locale *lattice* =
inf : *semlat*
sup : *semlat* **where** $leq = \lambda x \lambda y.\ inf.leq\ y\ x$

```
sig order = {
  this : tp
  leq  : tm (this ⇒ this ⇒ prop)
}
```

```
sig semlat = {
  this : tp
  struct ord : order = {this := this}
  inf  : tm (this ⇒ this ⇒ this)
}
```

```
sig lattice = {
  this : tp
  struct inf : semlat = {this := this}
  struct sup : semlat = {this := this, leq := λ[x] λ[y] inf.leq y x}
}
```

Functors between Type Classes

Assume

- a type class C with constant names $c_1, \ldots, c_m$ and axiom names $a_1, \ldots, a_n$
- type classes $C_1, \ldots, C_k$
- n-ary type operator t

Then:

$$
\begin{array}{l}
\textbf{instantiation } (\alpha_1, \ldots, \alpha_k)t :: (C_1, \ldots, C_k)C \textbf{ begin} \\
\quad c_1 = E_1 \; \ldots \; c_m = E_m \;\; P_1 \; \ldots \; P_n \\
\textbf{end}
\end{array}
$$

$$
\begin{array}{l}
\texttt{sig } \nu \; = \{ \\
\quad \texttt{struct } \alpha_1 \; : C_1 \\
\quad \vdots \\
\quad \texttt{struct } \alpha_k \; : C_k \\
\}
\end{array}
$$

$$
\begin{array}{llll}
\multicolumn{4}{l}{\texttt{view } \nu' \; : C \; \to \nu \; = \{} \\
\quad & \mathit{this} & := & t\ \alpha_1.\mathit{this}\ \ldots\ \alpha_k.\mathit{this} \\
& \ldots & & \\
& c_i & := & \ulcorner E_i \urcorner \\
& \ldots & & \\
& a_j & := & \ulcorner P_j \urcorner \\
& \ldots & & \\
\}
\end{array}
$$

Functor Applications

$$\textbf{instantiation}\ (\alpha_1,\ldots,\alpha_k)t :: (C_1,\ldots,C_k)C\ \textbf{begin}$$
$$c_1 = E_1\ \ldots\ c_m = E_m\ \ P_1\ \ldots\ P_n$$
$$\textbf{end}$$

$$\begin{array}{l} \texttt{sig}\ \nu\ = \{ \\ \quad \texttt{struct}\ \alpha_1\ : C_1 \\ \quad \vdots \\ \quad \texttt{struct}\ \alpha_k\ : C_k \\ \} \end{array}$$

$$\begin{array}{lll} \multicolumn{3}{l}{\texttt{view}\ \nu'\ : C\ \rightarrow \nu\ = \{} \\ \quad \mathit{this} & := & t\ \alpha_1.\mathit{this}\ \ldots\ \alpha_k.\mathit{this} \\ \quad \ldots & & \\ \quad c_i & := & \ulcorner E_i \urcorner \\ \quad \ldots & & \\ \quad a_j & := & \ulcorner P_j \urcorner \\ \quad \ldots & & \\ \} & & \end{array}$$

- Isabelle: if $t_i :: C_i$, then $(t_1,\ldots,t_k)t :: C$
- LF: if $\ulcorner t_i :: C_i \urcorner : C_i \rightarrow L$, then

$$\nu'\ \ \{\alpha_1 := \ulcorner t_1 :: C_1 \urcorner,\ \ldots,\ \alpha_k := \ulcorner t_k :: C_k \urcorner\} \quad : \quad C \rightarrow L$$

Adequacy

- $t :: C$ fully defined type class instance iff $\ulcorner \tau :: C \urcorner$ valid morphism out of $\ulcorner C \urcorner$ with $\ulcorner \tau :: C \urcorner(\mathit{this}) = \ulcorner t \urcorner$
- Subclass relation $C \subseteq D$ iff there is a morphism $\ulcorner D \urcorner \to \ulcorner C \urcorner$ in LF
- Accordingly for locales
- Isabelle theory T in restricted syntax valid iff LF signature $\ulcorner T \urcorner$ valid
- Extension to full Isabelle difficult
  - adequacy for elaboration of module system undesirable
  - fully formal definition of implemented system hard to get by

Conclusion

- Represented Isabelle in LF, module system covered

  good way to understand the primitives of Isabelle
- Presented alternative way to understand type classes

  also applicable to Haskell etc.
- Future work:
  - extend covered syntax
  - implement

  both very difficult