Isabelle — a generic theorem proving environment

Markus Wenzel

TU München

November 1998

Overview

1. The Meta-Logic: Isabelle/Pure
2. Simple set theory: Isabelle/HOL
3. Applications of Isabelle/HOL

Introduction

Church (1940): “The Simple Theory of Types”

Milner: “Interactive Tactical Proof Assistants”
(LCF $\approx$ 1970)

$\rightarrow$ many successors: HOL, Nuprl, Coq, PVS, ...

Isabelle (Paulson / Nipkow, established 1986)
provides a powerful generic framework for:

- rapid prototyping of deductive systems
- real working environments (object-logics), including
  HOL higher-order logic
  HOLCF domain theory
  FOL many-sorted first-order logic
  ZF set theory

$\rightarrow$ many applications world-wide

Isabelle's Meta Logic

Minimal fragment of Higher-Order Logic with equality.

Syntactic framework:

- λ-terms
- simple types (e.g. $bool$, $\alpha\ list$)
- ordered type classes (e.g. α_{ord} where $ord \sqsubset \top$)

Isabelle/Pure abstract syntax:

types

$prop$

$\alpha \to \beta$

consts

$\Rightarrow$	$:: prop \to prop \to prop$	**(infix)**
$\bigwedge$	$:: (\alpha_\top \to prop) \to prop$	**(binder)**
$\equiv$	$:: \alpha_\top \to \alpha_\top \to prop$	**(infix)**

where

$\varphi \Rightarrow \psi$	implication (logical entailment)
$\bigwedge x.\ \varphi[x]$	universal quantification
$t \equiv u$	equality

Isabelle/Pure: primitive rules

Implication:

$$\frac{\Gamma \vdash \psi}{\Gamma \setminus \{\varphi\} \vdash \varphi \Rightarrow \psi} \qquad \frac{\Gamma_1 \vdash \varphi \Rightarrow \psi \quad \Gamma_2 \vdash \varphi}{\Gamma_1, \Gamma_2 \vdash \psi}$$

Quantifier:

$$\frac{\Gamma \vdash \varphi[x]}{\Gamma \vdash \bigwedge x.\ \varphi[x]} \qquad \frac{\Gamma \vdash \bigwedge x.\ \varphi[x]}{\Gamma \vdash \varphi[t/x]}$$

Equivalence:

$$\frac{\Gamma_1 \vdash \varphi \Rightarrow \psi \quad \Gamma_2 \vdash \psi \Rightarrow \varphi}{\Gamma_1, \Gamma_2 \vdash \varphi \equiv \psi} \qquad \frac{\Gamma_1 \vdash \varphi \equiv \psi \quad \Gamma_2 \vdash \varphi}{\Gamma_1, \Gamma_2 \vdash \psi}$$

Equality:

$$\frac{}{\vdash t \equiv t} \qquad \frac{\Gamma_1 \vdash t \equiv u \quad \Gamma_2 \vdash u \equiv v}{\Gamma_1, \Gamma_2 \vdash t \equiv v} \qquad \frac{\Gamma \vdash t \equiv u}{\Gamma \vdash u \equiv t}$$

$$\frac{\Gamma_1 \vdash f \equiv g \quad \Gamma_2 \vdash t \equiv u}{\Gamma_1, \Gamma_2 \vdash f\ t \equiv g\ u} \qquad \frac{\Gamma \vdash t[x] \equiv u[x]}{\Gamma \vdash \lambda x.\ t[x] \equiv \lambda x.\ u[x]}$$

$$\frac{}{\vdash (\lambda x.\ t[x])\ u \equiv t[u/x]} \qquad \frac{\Gamma \vdash f\ x \equiv g\ x}{\Gamma \vdash f \equiv g}$$

Misc:

$$\frac{}{\varphi \vdash \varphi} \qquad \frac{\varphi \in Axms_\Theta}{\vdash \varphi} \qquad \frac{\Gamma \vdash \varphi[\alpha]}{\Gamma \vdash \varphi[\tau/\alpha]}$$

Isabelle/Pure: interesting rules

Main rules:

1. higher-order resolution (including lifting)
2. higher-order contextual rewriting

Resolution:

$$\text{(e)resolve} := \text{(e)res} \circ \text{inst}^* \circ (\Rightarrow\!\text{lift} \cup \textstyle\bigwedge\text{lift})^*$$

where:

$$\frac{\Gamma_1 \vdash \vec{\psi} \Rightarrow \varphi_i \quad \Gamma_2 \vdash \vec{\varphi}_1^{i-1} \varphi_i \vec{\varphi}_{i+1}^n \Rightarrow \varphi}{\Gamma_1, \Gamma_2 \vdash \vec{\varphi}_1^{i-1} \vec{\psi} \vec{\varphi}_{i+1}^n \Rightarrow \varphi} \text{ (res)}$$

$$\frac{\Gamma_1 \vdash \chi_j \vec{\psi} \Rightarrow \varphi_i \quad \Gamma_2 \vdash \vec{\varphi}_1^{i-1} (\vec{\chi}_1^{j-1} \chi_j \vec{\chi}_{j+1}^l \Rightarrow \varphi_i) \vec{\varphi}_{i+1}^n \Rightarrow \varphi}{\Gamma_1, \Gamma_2 \vdash \vec{\varphi}_1^{i-1} (\vec{\chi}_1^{j-1} \vec{\chi}_{j+1}^l \Rightarrow \vec{\psi}) \vec{\varphi}_{i+1}^n \Rightarrow \varphi} \text{ (eres)}$$

$$\frac{\Gamma \vdash \varphi[x]}{\Gamma \vdash \varphi[t/x]} \text{ (inst)}$$

$$\frac{\Gamma \vdash \vec{\varphi} \Rightarrow \varphi}{\Gamma \vdash (\chi \Rightarrow \vec{\varphi}) \Rightarrow (\chi \Rightarrow \varphi)} \text{ (}\Rightarrow\text{lift)}$$

$$\frac{\Gamma \vdash \vec{\varphi}[\vec{a}_\tau] \Rightarrow \varphi[\vec{a}_\tau]}{\Gamma \vdash \bigwedge x_\sigma.\ \vec{\varphi}[\vec{a}'_{\sigma\to\tau}\ x_\sigma/\vec{a}_\tau] \Rightarrow \bigwedge x_\sigma.\ \varphi[\vec{a}'_{\sigma\to\tau}\ x_\sigma/\vec{a}_\tau]} \text{ (}\textstyle\bigwedge\text{lift)}$$

Rules and proof states

Object-rules:

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi} \qquad \varphi \Rightarrow \psi \Rightarrow \varphi \wedge \psi$$

$$\frac{\begin{array}{c}[\varphi]\\ \vdots\\ \psi\end{array}}{\varphi \to \psi} \qquad (\varphi \Rightarrow \psi) \Rightarrow \varphi \to \psi$$

Proof states:

$$\frac{\begin{array}{c}[\vec{\chi}_1]\\ \vdots\\ \varphi_1\end{array} \quad \dots \quad \begin{array}{c}[\vec{\chi}_n]\\ \vdots\\ \varphi_n\end{array}}{\varphi} \qquad (\vec{\chi}_1 \Rightarrow \varphi_1) \Rightarrow \dots (\vec{\chi}_n \Rightarrow \varphi_n) \Rightarrow \varphi$$

Proof of φ: start with $\frac{\varphi}{\varphi}$, refine, finish with $\overline{\varphi}$.

basic steps:

- backward proof from subgoals (resolve)
- forward proof from assumptions (eresolve)
- trivial (assumption)

Note: higher-order hereditary Harrop formula, similar to λ-Prolog

Representing object-logics

Example: single sorted FOL

Abstract syntax:

types

i, o

consts

$\mathit{Trueprop} :: o \to \mathit{prop}$	**(implicit)**
$= :: i \to i \to o$	**(infix)**
$\neg :: o \to o$	**(prefix)**
$\wedge, \vee, \to, \leftrightarrow :: o \to o \to o$	**(infix)**
$\forall, \exists :: (i \to o) \to o$	**(binder)**

Object-rules: (natural deduction)

axioms

$$\to I : \ (P \Rightarrow Q) \Rightarrow (P \to Q)$$
$$\to E : \ (P \to Q) \Rightarrow P \Rightarrow Q$$

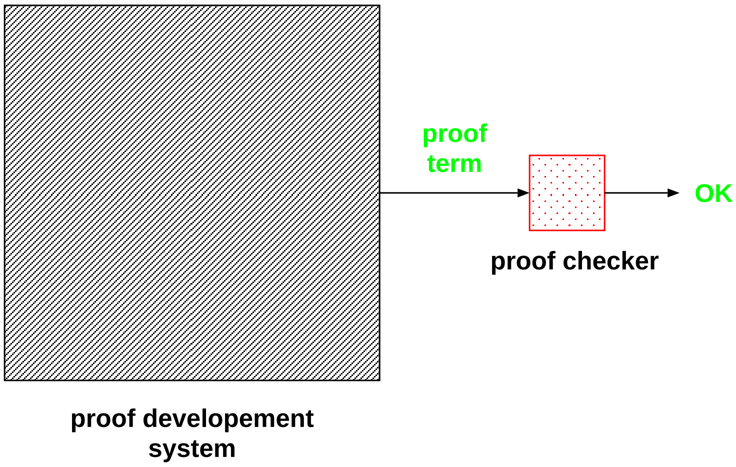
$$\forall I : \ (\bigwedge x.\ P\ x) \Rightarrow (\forall x.\ P\ x)$$
$$\forall E : \ (\forall x.\ P\ x) \Rightarrow (P\ a)$$

$$\exists I : \ (P\ a) \Rightarrow (\exists x.\ P\ x)$$
$$\exists E : \ (\exists x.\ P\ x) \Rightarrow (\bigwedge a.\ P\ a \Rightarrow Q) \Rightarrow Q$$

$\vdots$

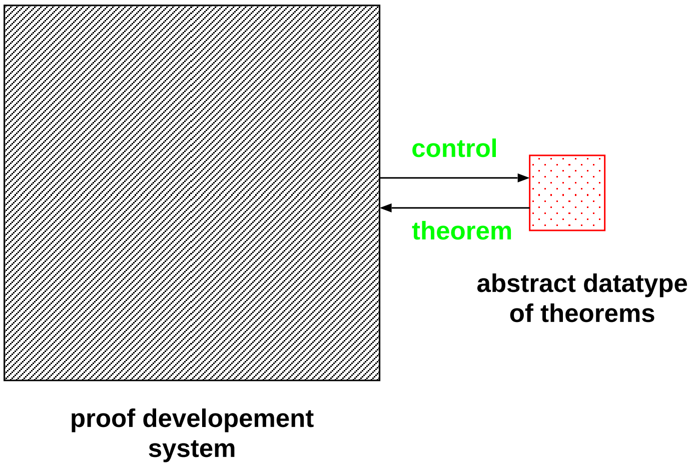
The Quest for Correctness

a) Explicit proof checking:



“de-Bruijn Principle”

b) Secure derivations:



“LCF system approach”

“Correctness by construction”

The Isabelle/Pure toolbox

Pure meta-logic + extra-logical mechanisms.

Generic tools:

- parser and pretty printer
- type unification (order-sorted)
- type inference
- higher-order unification
- higher-order resolution (intro / elim)
- tactics and tacticals
- automatic proof tools
- . . .

How to implement object-logics:

1. declare concrete syntax (mixfixes, translations)
2. declare abstract syntax (higher-order constants)
3. declare inference rules (meta-logical formulas)

→ *primitive inferences based on HO resolution*

4. instantiate generic proof tools (e.g. simplifier, classical tableau prover)
5. manually code special proof procedures

→ *semi-automated reasoning*

The HOL logic (Church / Gordon)

Why HOL?

$+$ typed language

$+$ simple deductive system

$+$ simple set theoretic semantics

$+$ quite expressive, may *define* substantial part of traditional mathematics

$-$ less expressive than full (untyped) set theory

$-$ simple types sometimes complicate formalisations

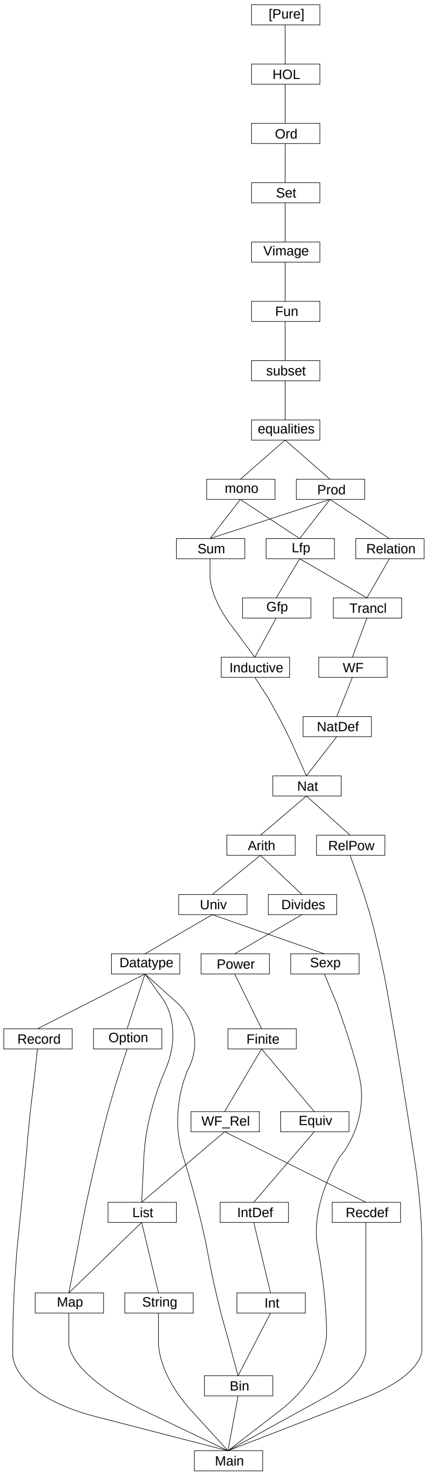
$\rightarrow$ HOL achieves a nice balance of *pure* vs. *applied*

Isabelle/HOL:

- directly embedded into the meta logic
- inherits the type system (including type classes)
- classical logic
- Hilbert's choice operator $\varepsilon\, x.\ P\ x$

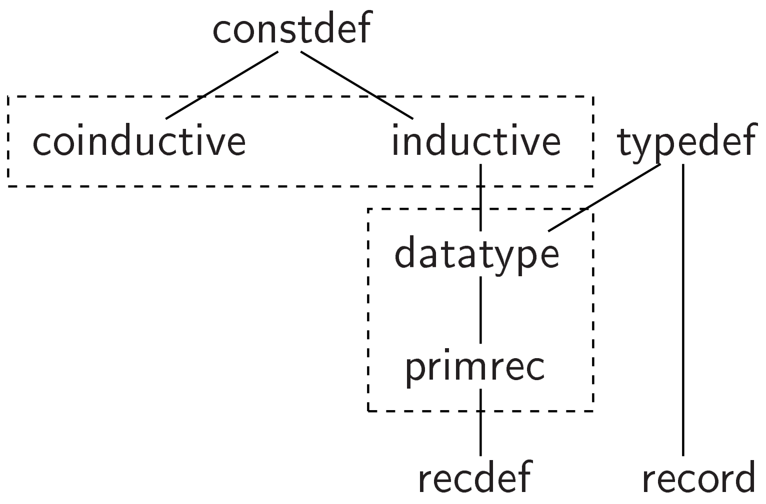
HOL is main workhorse for all projects at Munich.

Isabelle/HOL library



Isabelle/HOL packages

Definitional mechanisms:



constdef basic definitions, including overloading *(Pure primitive)*

typedef semantic subtypes *(HOL primitive)*

(co)inductive sets from types via Knaster-Tarski

datatype / primrec mutually and indirect (positive) inductive types

recdef general well-founded recursion, including pattern matching

record extensible records (single inheritance)

HOL/inductive

Input:

```
inductive  terms
  intrs    terms
  monos    theorems
```

Result:

- `intrs`: intro rules
- `induct`: (strong) induction rule
- `elim`: elimination rule — intro rules inversion
- `mk_cases`: create instantiated and simplified versions of `elim`

Additional features:

- mutually inductive sets
- disjunctive premises

HOL/datatype

Input:

`datatype` *type* `=`
constructors and types

Result:

- `distinct`, `inject`: freeness theorems
- `induct`: structural induction rule
- `exhaust`: case analysis rule
- `cases`: case simplifications
- `split`: contextual case split rule
- `size`: size function (for `recdef`)
- ...

Additional features:

- mutually inductive types
- indirect recursion (via datatypes only)

Example (1): expressions

```
(** Arithmetic expressions **)
datatype
  aexp = N nat
       | X loc
       | Op1 "nat => nat"        aexp
       | Op2 "nat => nat => nat" aexp aexp

(** Evaluation of arithmetic expressions **)
consts
 evala    :: "((aexp * state) * nat) set"
syntax
 "-a->"    :: "[aexp * state, nat] => bool"   (infixl 50)
translations
 "aesig -a-> n" == "(aesig,n) : evala"

inductive evala
  intrs
    N   "(N(n),s) -a-> n"
    X   "(X(x),s) -a-> s(x)"
    Op1 "(e,s) -a-> n ==> (Op1 f e,s) -a-> f(n)"
    Op2 "[| (e0,s) -a-> n0;  (e1,s)  -a-> n1 |]
           ==> (Op2 f e0 e1, s) -a-> f n0 n1"
```

Example (2): denotational semantics

```
consts
  A      :: aexp => state => nat
primrec
  "A(N(n))          = (%s. n)"
  "A(X(x))          = (%s. s(x))"
  "A(Op1 f a)       = (%s. f(A a s))"
  "A(Op2 f a0 a1) = (%s. f (A a0 s) (A a1 s))"


theorem "((a,s) -a-> n) = (A a s = n)"
```

HOL/recdef

Example:

```
consts gcd :: "nat * nat => nat"
recdef gcd "measure (%(x,y).x+y)"
  "gcd (0,y)           = y"
  "gcd (Suc x, 0)      = Suc x"
  "gcd (Suc x, Suc y) =
    (if (y <= x) then gcd(x - y, Suc y)
                 else gcd(Suc x, y - x))"
```

Isabelle at TU Munich

Main projects:

Isabelle/Bali (David von Oheimb, Cornelia Pusch)
Java and JVM formalization — type system, semantics, compilers

Isabelle/HOOL
(Wolfgang Naraschewski, Markus Wenzel)
Verification of OO programs — OO verification

Isabelle/IOA (Olaf Müller)
Verification environment for I/O Automata based on formalized meta-theory

Isabelle/IMP
(Tobias Nipkow, Leonor Prensa Nieto, et al)
Formalizing aspects of imperative programming languages — sequential and parallel

Isabelle/Isar (Markus Wenzel)
Intelligible semi-automated reasoning — readable formal proof documents

http://www.in.tum.de/~isabelle/