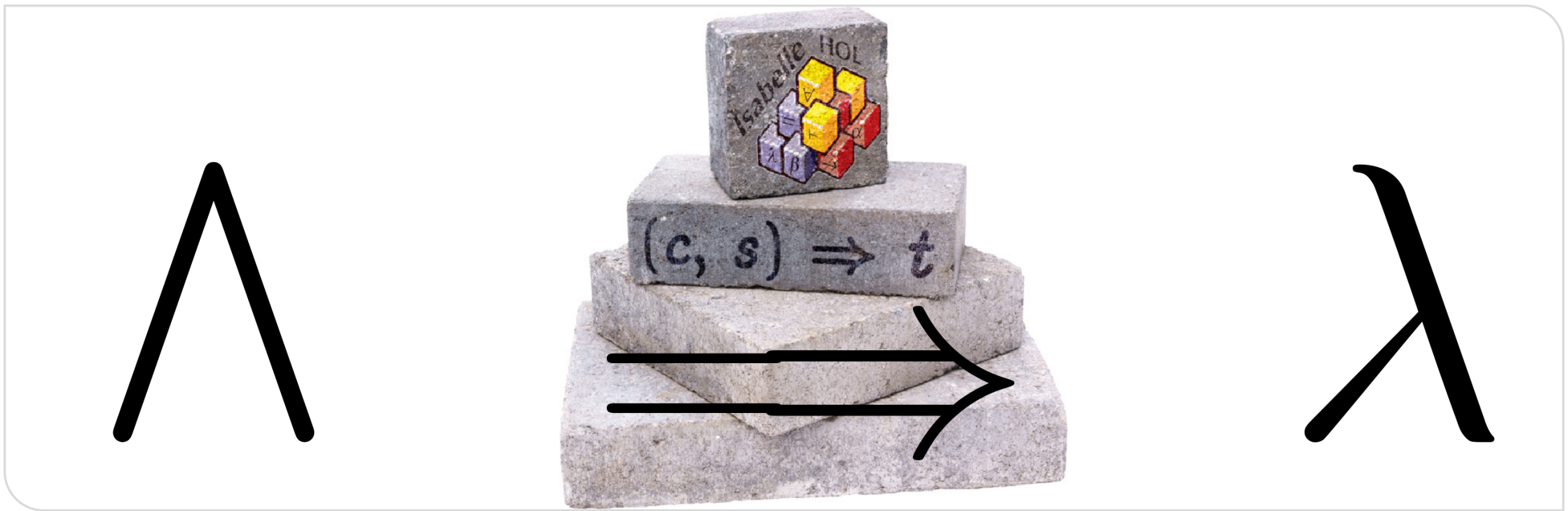
Karlsruhe Institute of Technology

KASTEL

Formal Systems II: Application

Higher-Order Logics and Interactive Theorem Proving with Isabelle II

Bernhard Beckert · Mattias Ulbrich · Michael Kirsten · Alexander Weigl | SS 2023



More on Function Definitions

Non-recursive definitions

- **definition** $sq ::\Rightarrow nat$ **where** $sqn = n * n$
- No pattern matching, just $f\, x_1 \ldots x_n = \ldots$

More on Function Definitions

Non-recursive definitions

- **definition** $sq ::\Rightarrow nat$ **where** $sqn = n * n$
- No pattern matching, just $f\, x_1 \ldots x_n = \ldots$

(Possibly) recursive functions: fun

- Pattern-matching over datatype constructors
- Order of equation matters
- Termination must be provable automatically by size measures
- Proves customized induction schema
- Example: **fun** $ack :: nat \Rightarrow nat \Rightarrow nat$ **where**

$$ack\ 0\ n = Suc\ n \mid$$
$$ack\ (Suc\ m)\ 0 = ack\ m\ (Suc\ 0) \mid$$
$$ack\ (Suc\ m)\ (Suc\ n) = ack\ m\ (ack\ (Suc\ m)\ n)$$

Proof method *simp*

Goal: 1. $[\![P_1; \ldots; P_m]\!] \Longrightarrow C$

apply$(simp\ add\text{:}\ eq_1\ \ldots\ eq_n)$

Simplify $P_1\ \ldots\ P_m$ and C using

- lemmas with attribute $simp$
- rules from **fun** and **datatype**
- additional lemmas $eq_1\ \ldots\ eq_n$
- assumptions $P_1\ \ldots\ P_m$

Variations:

- $(simp\ \ldots\ del\text{:}\ \ldots)$ removes $simp$-lemmas
- add and del are optional

auto versus *simp*

- *auto* acts on all subgoals
- *simp* acts only on subgoal 1
- *auto* applies *simp* and more
- *auto* can also be modified:
  $(auto\ simp\ add\colon \ldots\ simp\ del\colon \ldots)$

Rewriting with definitions

Definitions (**definition**) must be used explicitly:

$$(simp\ add\text{: } f\_def \ldots)$$

f is the function whose definition is to be unfolded.

Case splitting with *simp/auto*

Automatic:

$$P\ (\mathit{if}\ A\ \mathit{then}\ s\ \mathit{else}\ t)$$
$$=$$
$$(A \longrightarrow P(s)) \wedge (\neg A \longrightarrow P(t))$$

By hand:

$$P\ (\mathit{case}\ e\ \mathit{of}\ 0 \Rightarrow a \mid \mathit{Suc}\ n \Rightarrow b)$$
$$=$$
$$(e = 0 \longrightarrow P(a)) \wedge (\forall n.\ e = \mathit{Suc}\ n \longrightarrow P(b))$$

Proof method: *(simp split: nat.split)*
Or *auto*. Similar for any datatype *t*: *t.split*

More Proof Methods

intro $\langle intro\text{-}rules \rangle$	repeatedly applies intro rules
elim $\langle elim\text{-}rules \rangle$	repeatedly applies elim rules
clarify	applies all safe rules that do not split the goal
safe	applies all safe rules
blast	an automatic tableaux prover (works well on predicate logic)
fast	another automatic search tactic
fastforce	rewriting, logic, sets, relations and a bit of arithmetic
arith	proves linear formulas (no “$*$”), complete for quantifier-free real and Presburger arithmetic

Download

https://www21.in.tum.de/teaching/fds/SS22/assets/Demos/Simp_Demo.thy

Back to HOL

Base: $bool, \Rightarrow, ind \qquad =, \longrightarrow, \varepsilon$

And the rest is definitions:

$$
\begin{array}{lcl}
\text{True} & \equiv & (\lambda x :: bool.\ x) = (\lambda x.\ x) \\
\text{All } P & \equiv & P = (\lambda x.\ \text{True}) \\
\text{Ex } P & \equiv & \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q \\
\text{False} & \equiv & \forall P.\ P \\
\neg P & \equiv & P \longrightarrow \text{False} \\
P \wedge Q & \equiv & \forall R.\ (P \longrightarrow Q \longrightarrow R) \longrightarrow R \\
P \vee Q & \equiv & \forall R.\ (P \longrightarrow R) \longrightarrow (Q \longrightarrow R) \longrightarrow R \\
\text{If } P\ x\ y & \equiv & \text{SOME } z.\ (P = \text{True} \longrightarrow z = x) \wedge (P = \text{False} \longrightarrow z = y) \\
\text{inj } f & \equiv & \forall x\ y.\ f\ x = f\ y \longrightarrow x = y \\
\text{surj } f & \equiv & \forall y.\ \exists x.\ y = f\ x
\end{array}
$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s = t \quad P\ s}{P\ t}\ \text{subst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

$$\frac{P \Longrightarrow Q}{P \longrightarrow Q}\ \text{impl} \qquad \frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$

$$\frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)}\ \text{iff}$$

$$\frac{}{P = \text{True} \vee P = \text{False}}\ \text{True\_or\_False}$$

$$\frac{P\ ?x}{P\ (\text{SOME}\ x.\ P\ x)}\ \text{somel}$$

$$\frac{}{\exists f :: ind \Rightarrow ind.\ \text{inj}\ f \wedge \neg \text{surj}\ f}\ \text{infty}$$

That's it.

- ➜ 3 basic constants
- ➜ 3 basic types
- ➜ 9 axioms

With this you can define and derive all the rest.

Isabelle knows 2 more axioms:

$$\frac{x = y}{x \equiv y}\ \text{eq\_reflection} \qquad \frac{}{(\text{THE } x.\ x = a) = a}\ \text{the\_eq\_trivial}$$

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

Phrase theorems like this $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

not like this $A_1 \wedge \ldots \wedge A_n \longrightarrow A$

Sets over type $'a$

$'a\ set$

- $\{\}, \quad \{e_1,\ldots,e_n\}$
- $e \in A, \quad A \subseteq B$
- $A \cup B, \quad A \cap B, \quad A - B, \quad - A$
- $\ldots$

$\in$	`\<in>`	`:`
$\subseteq$	`\<subseteq>`	`<=`
$\cup$	`\<union>`	`Un`
$\cap$	`\<inter>`	`Int`

Set comprehension

- $\{x.\ P\}$ where x is a variable
- But not $\{t.\ P\}$ where t is a proper term
- Instead: $\{t\ |x\ y\ z.\ P\}$
  is short for $\{v.\ \exists\, x\ y\ z.\ v = t \wedge P\}$
  where x, y, z are the free variables in t

Isar core syntax

proof = **proof** [method] step* **qed**
| **by** method

method = $(simp \ldots) \mid (blast \ldots) \mid (induction \ldots) \mid \ldots$

step = **fix** variables $(\bigwedge)$
| **assume** prop $(\Longrightarrow)$
| [**from** fact+] (**have** | **show**) prop proof

prop = [name:] "formula"

fact = name | ...

Example: Cantor's theorem

lemma $\neg$ $surj(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume $surj$, show $False$
  assume a: $surj\ f$
  from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
    by$(simp\ add$: $surj\_def)$
  from b **have** c: $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$
    by $blast$
  from c **show** $False$
    by $blast$
qed

Download

https://www21.in.tum.de/teaching/fds/SS22/assets/Demos/Isar_Demo.thy

Abbreviations

$this$	=	the previous proposition proved or assumed
then	=	**from** $this$
thus	=	**then show**
hence	=	**then have**

using and **with**

(**have**|**show**) prop **using** facts

=

from facts (**have**|**show**) prop

with facts

=

from facts *this*

Structured lemma statement

lemma
  fixes $f :: 'a \Rightarrow 'a\ set$
  assumes s: $surj\ f$
  shows $False$
proof $-$ no automatic proof step
  have $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$ **using** s
    by$(auto\ simp: surj\_def)$
  thus $False$ **by** $blast$
qed

Proves $surj\ f \Longrightarrow False$
but $surj\ f$ *becomes local fact* s *in proof.*

The essence of structured proofs

Assumptions and intermediate facts
can be named and referred to explicitly and selectively

Structured lemma statements

fixes $x :: \tau_1$ **and** $y :: \tau_2$ ...
assumes *a:* P **and** *b:* Q ...
shows R

- **fixes** and **assumes** sections optional
- **shows** optional if no **fixes** and **assumes**

Case distinction

show R
proof $cases$
  assume P
  $\vdots$
  show R $\langle proof \rangle$
next
  assume $\neg\ P$
  $\vdots$
  show R $\langle proof \rangle$
qed

have $P \vee Q$ $\langle proof \rangle$
then show R
proof
  assume P
  $\vdots$
  show R $\langle proof \rangle$
next
  assume Q
  $\vdots$
  show R $\langle proof \rangle$
qed

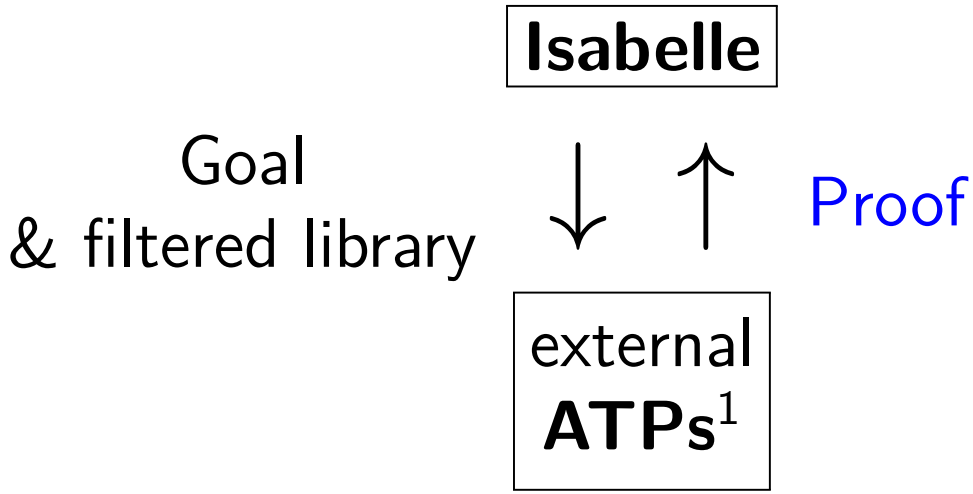
Set equality and subset

show $A = B$
proof
  show $A \subseteq B$ $\langle proof \rangle$
next
  show $B \subseteq A$ $\langle proof \rangle$
qed

show $A \subseteq B$
proof
  fix x
  assume $x \in A$
  $\vdots$
  show $x \in B$ $\langle proof \rangle$
qed

Sledgehammer

Architecture:



Characteristics:

- Sometimes it works,
- sometimes it doesn't.

Do you feel lucky?

[1] Automatic Theorem Provers

Sledgehammer

Sledgehammer:

- ➜ *Connects Isabelle with ATPs and SMT solvers:*
  E, SPASS, Vampire, CVC3, Yices, Z3
- ➜ *Simple invocation:*
  - ➜ *Users don't need to select or know facts*
  - ➜ *or ensure the problem is first-order*
  - ➜ *or know anything about the automated prover*
- ➜ *Exploits local parallelism and remote servers*

Download

https://www21.in.tum.de/teaching/fds/SS22/assets/Demos/Auto_Proof_Demo.thy

Download `Chapter2.thy` from

http://concrete-semantics.org/Exercises/templates.tar

and try to do some more exercises.

Other Facets of Isabelle

- Document preparation: you can generate LaTeX documents from your theories
- Axiomatic type classes: a general approach to polymorphism and overloading when there are shared laws
- Code generation: you can generate executable code from the formal functional programs you have verified
  $\Rightarrow$ Algorithms can be verified and then executed (ML, Haskell, Scala)
- Locales: encapsulated contexts, ideal for formalising abstract mathematics