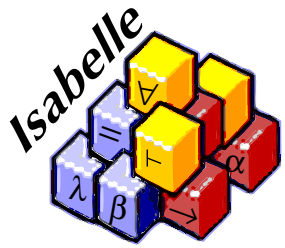
Proof terms in Isabelle

Stefan Berghofer



Isabelle — a generic theorem prover

Meta logic

Intuitionistic higher order logic

- Terms: $t = x \mid c \mid \lambda x :: \tau.\ t \mid t\ t$
- Types: $\tau = \alpha \mid (\tau_1, \ldots, \tau_n)tc$ where $tc \in \{\rightarrow,\ \text{prop},\ \ldots\}$
  first order, Hindley-Milner polymorphism
- Logical connectives:

universal quantification	$\bigwedge$	::	$(\alpha \rightarrow \text{prop}) \rightarrow \text{prop}$
implication	$\Longrightarrow$	::	$\text{prop} \rightarrow \text{prop} \rightarrow \text{prop}$
equality	$\equiv$	::	$\alpha \rightarrow \alpha \rightarrow \text{prop}$

Object logics

are formalized using meta logic

Formalizing object logics

$$\begin{array}{lll} \mathsf{Tr} & :: & \mathsf{bool} \to \mathsf{prop} \\ \longrightarrow & :: & \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} \\ \forall & :: & (\alpha \to \mathsf{bool}) \to \mathsf{bool} \\ \exists & :: & (\alpha \to \mathsf{bool}) \to \mathsf{bool} \end{array}$$

$$\begin{array}{lll} \mathsf{impl} & : & \bigwedge A\ B.\ (\mathsf{Tr}\ A \Longrightarrow \mathsf{Tr}\ B) \Longrightarrow \mathsf{Tr}\ (A \longrightarrow B) \\ \mathsf{mp} & : & \bigwedge P\ Q.\ \mathsf{Tr}\ (P \longrightarrow Q) \Longrightarrow \mathsf{Tr}\ P \Longrightarrow \mathsf{Tr}\ Q \\ \mathsf{allI} & : & \bigwedge P.\ (\bigwedge x.\ \mathsf{Tr}\ (P\ x)) \Longrightarrow \mathsf{Tr}\ (\forall x.\ P\ x) \\ \mathsf{spec} & : & \bigwedge P\ x.\ \mathsf{Tr}\ (\forall x.\ P\ x) \Longrightarrow \mathsf{Tr}\ (P\ x) \\ \mathsf{exI} & : & \bigwedge P\ x.\ \mathsf{Tr}\ (P\ x) \Longrightarrow \mathsf{Tr}\ (\exists x.\ P\ x) \\ \mathsf{exE} & : & \bigwedge P\ Q.\ \mathsf{Tr}\ (\exists x.\ P\ x) \Longrightarrow (\bigwedge x.\ \mathsf{Tr}\ (P\ x) \Longrightarrow \mathsf{Tr}\ Q) \Longrightarrow \mathsf{Tr}\ Q \end{array}$$

Proofs

- User-level view of theorem provers:
  - Description of proof search using expressive tactic languages
  - Interpreter for "readable" proof documents (Isar [Wenzel, 2002])

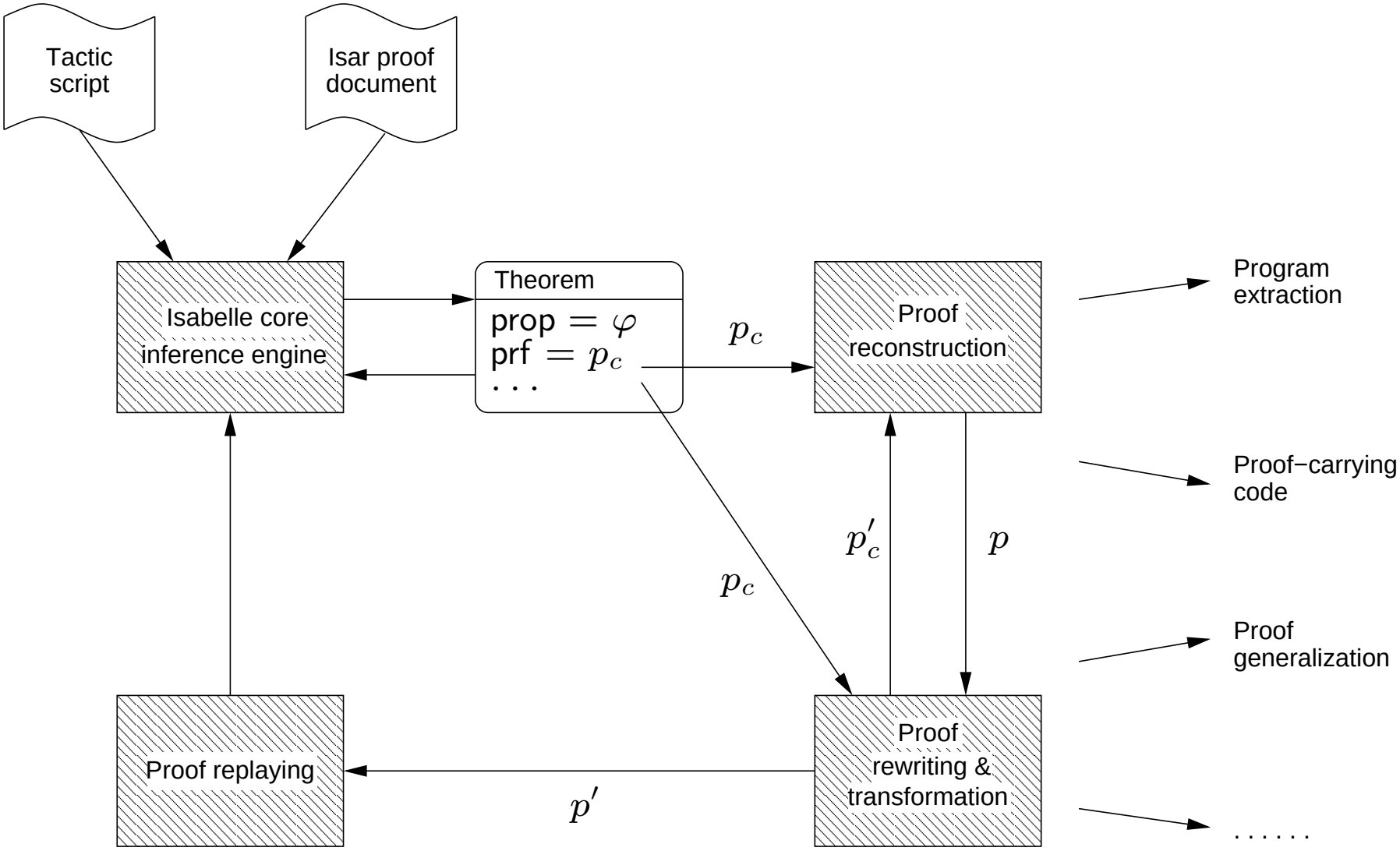
  $\leadsto$ Problem: Checking requires complex machinery

- **Requirement:** simple, independently checkable proofs
  (Automath [de Bruijn, 1970], CVC [Stump / Dill, 2002])

  $\leadsto$ more primitive proof format needed

- Applications:
  - Proof-Carrying Code [Wildmoser, 2006]
  - Exchange of proofs with other systems [McLaughlin, 2006]
  - Proof generalization and reuse [Johnsen / Lüth, 2004]
  - Program extraction
  - "Proof Mining"

Architecture



Proof representation

Proofs as λ-terms

p, q	$=$	h	Hypothesis
	$\mid$	$c_{\{\overline{\alpha} \mapsto \overline{\tau}\}}$	Proof constant (reference to axiom / theorem)
	$\mid$	$p \cdot t$	$\bigwedge$ -elimination
	$\mid$	$p \bullet q$	$\Longrightarrow$ -elimination
	$\mid$	$\boldsymbol{\lambda} x :: \tau.\ p$	$\bigwedge$ -introduction
	$\mid$	$\boldsymbol{\lambda} h : \varphi.\ p$	$\Longrightarrow$ -introduction

Proof checking

$$\frac{}{\Gamma, h : t, \Gamma' \vdash h : t} \qquad \frac{\Sigma(c) = \varphi}{\Gamma \vdash c_{\{\overline{\alpha} \mapsto \overline{\tau}\}} : \varphi\{\overline{\alpha} \mapsto \overline{\tau}\}}$$

$$\frac{\Gamma \vdash p : \bigwedge x :: \tau.\ \varphi \quad \Gamma \vdash t :: \tau}{\Gamma \vdash p \cdot t : P\{x \mapsto t\}} \qquad \frac{\Gamma, x :: \tau \vdash p : \varphi}{\Gamma \vdash \boldsymbol{\lambda} x :: \tau.\ p : \bigwedge x :: \tau.\ \varphi}$$

$$\frac{\Gamma \vdash p : \varphi \Longrightarrow \psi \quad \Gamma \vdash q : \varphi}{\Gamma \vdash p \bullet q : \psi} \qquad \frac{\Gamma, h : \varphi \vdash p : \psi \quad \Gamma \vdash \varphi :: \mathsf{prop}}{\Gamma \vdash \boldsymbol{\lambda} h : \varphi.\ p : \varphi \Longrightarrow \psi}$$

Proofs and theorems in ML

```
datatype thm = Thm of
 {thy_ref: theory_ref,          (*reference to theory*)
  der: bool * proof,            (*derivation*)
  prop: term,                   (*conclusion*)
  hyps: term list,              (*hypotheses as ordered list*)
  shyps: sort list,             (*sort hypotheses as ordered list*)
  ...};
datatype proof =
   PBound of int
 | Abst of string * typ option * proof
 | AbsP of string * term option * proof
 | % of proof * term option
 | %% of proof * proof
 | Hyp of term
 | PThm of string * proof * term * typ list option
 | ...;
```

Proof synthesis

Resolution

$$\dfrac{\dfrac{\psi_1 \quad \cdots \quad \psi_m}{\boxed{\begin{array}{c}\psi \\ =^{?} \\ \varphi_1\end{array}}}\, r \qquad \varphi_2 \qquad \cdots \qquad \varphi_n}{\varphi}\, s \quad \longmapsto \quad \theta\left(\dfrac{\psi_1 \quad \cdots \quad \psi_m \quad \varphi_2 \quad \cdots \quad \varphi_n}{\varphi}\right)$$

where $\theta(\psi) = \theta(\varphi_1)$

Resolution = function composition

$$\theta(\boldsymbol{\lambda}(h_1 : \psi_1) \ \ldots \ (h_m : \psi_m).\ s\ (r\ h_1\ \cdots\ h_m))$$

Initial proof state = identity function

$$\boldsymbol{\lambda} h : \varphi.\ h$$

Proof synthesis – continued

Lifting over assumptions

$$\frac{P_1 \quad \ldots \quad P_m}{C}\, R \qquad \longmapsto \qquad \frac{\overline{Q_n} \Longrightarrow P_1 \quad \ldots \quad \overline{Q_n} \Longrightarrow P_m}{\overline{Q_n} \Longrightarrow C}$$

proof term:

$$\lambda \overline{r_m}\ \overline{q_n}.\ R\ (\overline{r_m\ \overline{q_n}})$$

Lifting over parameters

$$\frac{P_1\,[\overline{a_k}] \quad \ldots \quad P_m\,[\overline{a_k}]}{C\,[\overline{a_k}]}\, R\,[\overline{a_k}] \qquad \longmapsto \qquad \frac{\bigwedge \overline{x_n}.\ P_1\left[\overline{a'_k\ \overline{x_n}}\right] \quad \ldots \quad \bigwedge \overline{x_n}.\ P_m\left[\overline{a'_k\ \overline{x_n}}\right]}{\bigwedge \overline{x_n}.\ C\left[\overline{a'_k\ \overline{x_n}}\right]}$$

proof term:

$$\lambda \overline{r_m}\ \overline{x_n}.\ R\left[\overline{a'_k\ \overline{x_n}}\right]\ \ (\overline{r_m\ \overline{x_n}})$$

Inference rules in ML

```
fun assume (Cterm {t = prop, T, ...}) =
  Thm {der = (false, Hyp prop),
    prop = prop,
    hyps = [prop], ...};

fun implies_elim (Thm {der = (ora, prf), prop, hyps, ...})
    (Thm {der = (oraA, derA), prop = propA, hyps = hypsA, ...})
    case prop of
      Const ("==>", _) $ A $ B =>
        if A aconv propA then
          Thm {der = (ora orelse oraA, prf %% prfA),
            prop = B,
            hyps = union_hyps hypsA hyps, ...}
        else raise THM "bad proposition"
    | _ => raise THM "bad proposition";
```

Inference rules in ML – continued

```
fun implies_intr (Cterm {t = A, T, ...})
    (Thm {der = (ora, prf), prop, hyps, ...}) =
  if T <> propT then raise THM "assumptions must have type prop"
  else Thm {der = (ora, AbsP ("H", NONE, abshyph 0 A prf)),
    prop = implies $ A $ prop,
    hyps = remove_hyps A hyps, ...};
```

```
fun abshyp i h prf = (abshyp' i h prf handle SAME => prf)
and abshyp' i h (Hyp t) = if h aconv t then PBound i else raise SAME
  | abshyp' i h (Abst (s, T, prf)) = Abst (s, T, abshyp' i h prf)
  | abshyp' i h (AbsP (s, t, prf)) = AbsP (s, t, abshyp' (i+1) h prf)
  | abshyp' i h (prf % t) = abshyp' i h prf % t
  | abshyp' i h (prf1 %% prf2) = (abshyp' i h prf1 %% abshyp i h prf2
      handle SAME => prf1 %% abshyp' i h prf2)
  | abshyp' _ _ _ = raise SAME
```

Proof compression

Problem: proofs contain syntactic redundancies:

$$
\begin{array}{l}
\mathit{impI} \cdot A \vee B \cdot B \vee A \cdot \\
\quad (\boldsymbol{\lambda} H\colon A \vee B. \\
\qquad \mathit{disjE} \cdot A \cdot B \cdot B \vee A \cdot H \cdot (\mathit{disjI2} \cdot A \cdot B) \cdot (\mathit{disjI1} \cdot B \cdot A))
\end{array}
$$

Solution: placeholders for terms

$$
\begin{array}{l}
\mathit{impI} \cdot \_ \cdot \_ \cdot \\
\quad (\boldsymbol{\lambda} H\colon \_. \\
\qquad \mathit{disjE} \cdot \_ \cdot \_ \cdot \_ \cdot H \cdot (\mathit{disjI2} \cdot \_ \cdot \_) \cdot (\mathit{disjI1} \cdot \_ \cdot \_))
\end{array}
$$

- Reconstruction of missing information by solving constraints between terms using unification
- Similar to type inference in functional programming languages
- Quantifier rules require higher-order unification
  - Undecidable in general [G. Huet, 1974]
  - Pattern unification [D. Miller, 1991]: decidable, most general unifiers
- See also: implicit arguments in Coq / Lego

Proof compression strategies

Goal: only omit information that can be reconstructed using pattern unification.

- Dynamic: analyze data flow in proof ($\rightsquigarrow$ "bidirectional" algorithms)
- Static: analyze signature, i.e. "types" of inference rules

impI : $\bigwedge P\ Q.\quad (P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$
mp : $\bigwedge P\ Q.\quad P \longrightarrow Q \Longrightarrow P \Longrightarrow Q$
allI : $\bigwedge P.\quad (\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$
spec : $\bigwedge P\ x.\quad \forall x.\ P\ x \Longrightarrow P\ x$
exI : $\bigwedge P\ x.\quad P\ x \Longrightarrow \exists x.\ P\ x$
exE : $\bigwedge P\ Q.\quad \exists x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q) \Longrightarrow Q$

We may omit a parameter of an inference rule if it ...

- ... only occurs with distinct bound variables as arguments, and ...
- ... does not occur as argument of another parameter

$\rightsquigarrow$ "strict occurrences" in Twelf [Pfenning, Schürmann]

Reconstructing omitted information

Reconstruction judgement: $\Gamma \vdash p_p \rhd (p,\ \varphi,\ C)$

p_p	Proof term with placeholders
p	Proof term with placeholders filled in
φ	Proposition proved by p
C	Constraints that must be satisfied in order for p to be a proof of φ

Reconstruction rules ($\approx$ type inference)

$$\frac{\Gamma, h : ?f_{\overline{\tau_\Gamma} \Rightarrow \mathsf{prop}}\ \overline{V_\Gamma} \vdash p_p \rhd (p,\ \psi,\ C)}{\Gamma \vdash (\boldsymbol{\lambda} h : \_.\ p_p) \rhd ((\boldsymbol{\lambda} h : ?f_{\overline{\tau_\Gamma} \Rightarrow \mathsf{prop}}\ \overline{V_\Gamma}.\ p),\ ?f_{\overline{\tau_\Gamma} \Rightarrow \mathsf{prop}}\ \overline{V_\Gamma} \Longrightarrow \psi,\ C)}\ \mathsf{ImpI_p}$$

$$\frac{\Gamma \vdash p_p \rhd (p,\ \varphi,\ C) \quad \Gamma \vdash q_p \rhd (q,\ \psi,\ D)}{\Gamma \vdash (p_p \cdot q_p) \rhd ((p \cdot q),\ ?f_{\overline{\tau_\Gamma} \Rightarrow \mathsf{prop}}\ \overline{V_\Gamma},\ \{\varphi =^{?} (\psi \Longrightarrow ?f_{\overline{\tau_\Gamma} \Rightarrow \mathsf{prop}}\ \overline{V_\Gamma})\} \cup C \cup D)}\ \mathsf{ImpE}$$

Proof size

theorem name	uncompressed		compressed	
	terms	size	size	ratio (%)
IntArith.pos-zmult-eq-1-iff	49046	247726	5615	97.73
IntDiv.pos-zdiv-mult-2	46985	245920	6218	97.47
IntDiv.pos-zmod-mult-2	33013	175819	4457	97.47
Presburger.int-ge-induct	29603	206508	4441	97.85
List.nibble.split	27938	114541	1792	98.44
IntDiv.self-quotient	27310	149712	3722	97.51
IntArith.int-le-induct	25381	191215	4055	97.88
IntArith.int-ge-induct	24962	187436	3962	97.89
List.list-all2-append1	23900	116870	2821	97.59
IntDiv.divAlg-correct	23886	117037	2892	97.53
List.nibble.split-asm	23762	116263	1808	98.44
List.nth-upt	21721	101355	2454	97.58
IntDiv.zminus1-lemma	18485	99613	3082	96.91
IntDiv.zadd1-lemma	17716	94929	3554	96.26
IntDiv.quorem-neg	17329	91425	2678	97.07

Proof simplification

$$(mp \cdot A \cdot B \cdot (impI \cdot A \cdot B \cdot prf)) \equiv prf$$

$$(impI \cdot A \cdot B \cdot (mp \cdot A \cdot B \cdot prf)) \equiv prf$$

$$(spec \cdot TYPE(\alpha) \cdot P \cdot x \cdot (allI \cdot TYPE(\alpha) \cdot P \cdot prf)) \equiv prf \cdot x$$

$$(allI \cdot TYPE(\alpha) \cdot P \cdot (\boldsymbol{\lambda} x :: \alpha.\ spec \cdot TYPE(\alpha) \cdot P \cdot x \cdot prf)) \equiv prf$$

$$(exE \cdot TYPE(\alpha) \cdot P \cdot Q \cdot (exI \cdot TYPE(\alpha) \cdot P \cdot x \cdot prf_1) \cdot prf_2) \equiv (prf_2 \cdot x \cdot prf_1)$$

$$(exE \cdot TYPE(\alpha) \cdot P \cdot Q \cdot prf \cdot (exI \cdot TYPE(\alpha) \cdot P)) \equiv prf$$

$$(disjE \cdot P \cdot Q \cdot R \cdot (disjI_1 \cdot P \cdot Q \cdot prf_1) \cdot prf_2 \cdot prf_3) \equiv (prf_2 \cdot prf_1)$$

$$(disjE \cdot P \cdot Q \cdot R \cdot (disjI_2 \cdot Q \cdot P \cdot prf_1) \cdot prf_2 \cdot prf_3) \equiv (prf_3 \cdot prf_1)$$

$$(conjunct_1 \cdot P \cdot Q \cdot (conjI \cdot P \cdot Q \cdot prf_1 \cdot prf_2)) \equiv prf_1$$

$$(conjunct_2 \cdot P \cdot Q \cdot (conjI \cdot P \cdot Q \cdot prf_1 \cdot prf_2)) \equiv prf_2$$

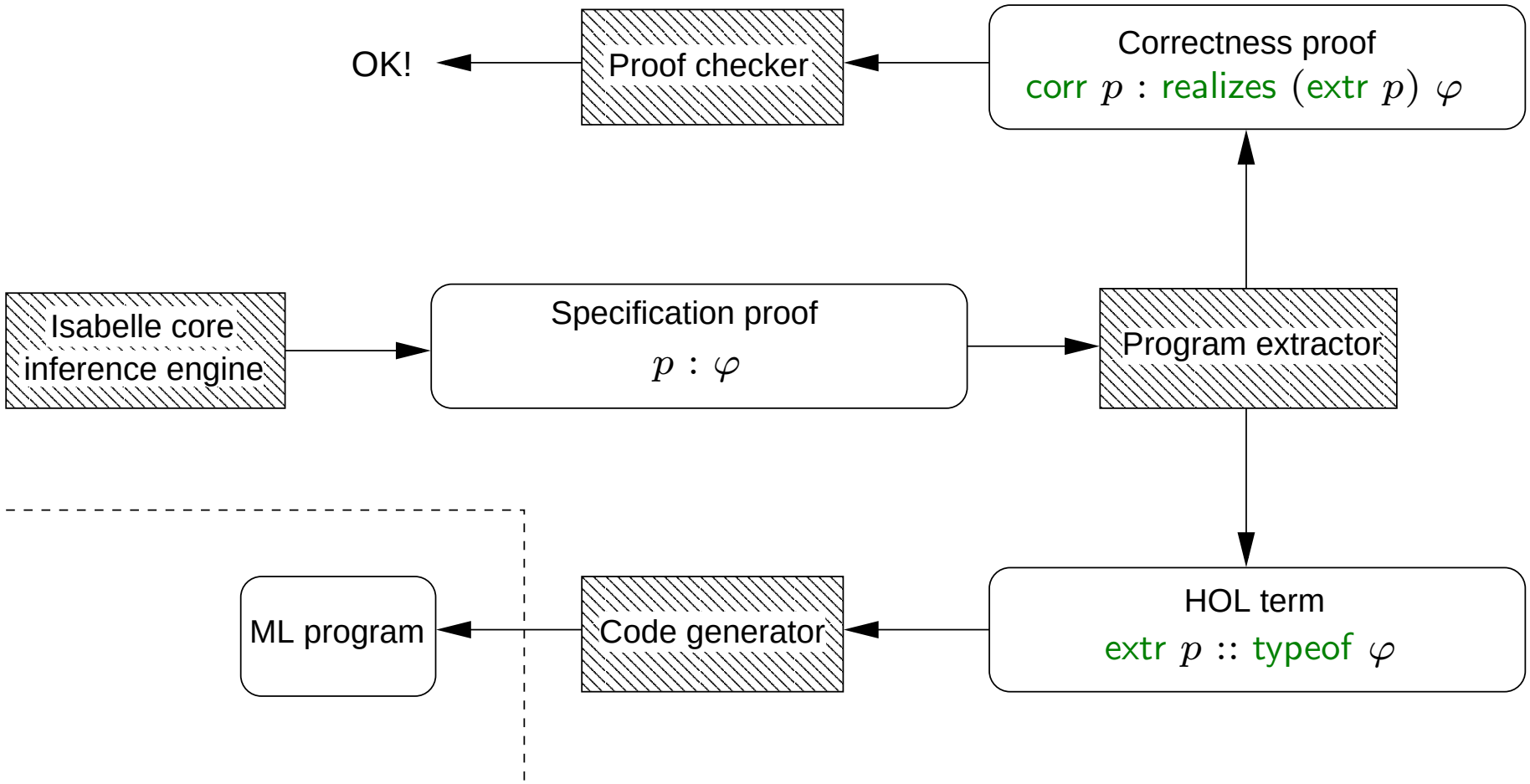
$$(iffD_1 \cdot A \cdot B \cdot (iffI \cdot A \cdot B \cdot prf_1 \cdot prf_2)) \equiv prf_1$$

$$(iffD_2 \cdot A \cdot B \cdot (iffI \cdot A \cdot B \cdot prf_1 \cdot prf_2)) \equiv prf_2$$

Application: Program extraction

- Based on modified realizability [Kleene, Kreisel]
- Internal extraction: Programs extracted from HOL proofs are HOL functions
- Use constructive fragment of HOL for proving computationally relevant statements.
- Computationally irrelevant statements may also be proved using classical reasoning
- **Case studies:**
  - Higman's lemma (also available as a Coq contribution, thanks to Pierre Letouzey)
  - Normalization algorithm for simply-typed λ-calculus
  - Euclid's theorem: There are infinitely many primes ("romantic proof" by Freek)
  - Pigeonhole principle
  - Warshall algorithm
  - Quotient and remainder
- **TODO:**
  - Program extraction from classical proofs (what approach?)
  - Program extraction for other logics (e.g. CZF)

A framework for program extraction



From proofs to programs — (1) Readable Isar proof document

theorem $division$: $\exists\, r\ q.\ a = Suc\ b * q + r \wedge r \leq b$
proof $(induct\ a)$
  case 0
  have $0 = Suc\ b * 0 + 0 \wedge 0 \leq b$ **by** $simp$
  thus $?case$ **by** $rules$
next
  case $(Suc\ a)$
  then obtain $r\ q$ **where** I: $a = Suc\ b * q + r$ **and** $r \leq b$ **by** $rules$
  from $nat\text{-}eq\text{-}dec$ **show** $?case$
  proof
    assume $r = b$
    with I **have** $Suc\ a = Suc\ b * (Suc\ q) + 0 \wedge 0 \leq b$ **by** $simp$
    thus $?case$ **by** $rules$
  next
    assume $r \neq b$ **hence** $r < b$ **by** $(simp\ add\text{:}\ order\text{-}less\text{-}le)$
    with I **have** $Suc\ a = Suc\ b * q + (Suc\ r) \wedge (Suc\ r) \leq b$ **by** $simp$
    thus $?case$ **by** $rules$
  qed
qed

From proofs to programs — (2) Primitive proof object

$$
\begin{array}{l}
\mathsf{nat.induct} \cdot (\lambda u.\ \exists r\ q.\ u = \mathsf{Suc}\ b * q + r \wedge r \leq b) \cdot a \cdot \\
\quad (\mathsf{exI} \cdot (\lambda r.\ \exists q.\ 0 = \mathsf{Suc}\ b * q + r \wedge r \leq b) \cdot 0 \cdot \\
\qquad (\mathsf{exI} \cdot (\lambda q.\ 0 = \mathsf{Suc}\ b * q + 0 \wedge 0 \leq b) \cdot 0 \cdot \\
\qquad\quad (\mathsf{conjI} \cdot 0 = \mathsf{Suc}\ b * 0 + 0 \cdot 0 \leq b \cdot \ldots \cdot \ldots))) \cdot \\
\quad (\boldsymbol{\lambda} n\ H : \exists r\ q.\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b. \\
\qquad \mathsf{exE} \cdot (\lambda r.\ \exists q.\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b) \cdot \exists r\ q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b \cdot H \cdot \\
\qquad\quad (\boldsymbol{\lambda} r\ H : \exists q.\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b. \\
\qquad\qquad \mathsf{exE} \cdot (\lambda q.\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b) \cdot \exists r\ q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b \cdot H \cdot \\
\qquad\qquad\quad (\boldsymbol{\lambda} q\ H : n = \mathsf{Suc}\ b * q + r \wedge r \leq b. \\
\qquad\qquad\qquad \mathsf{disjE} \cdot r = b \cdot r \neq b \cdot \exists r\ q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b \cdot \\
\qquad\qquad\qquad\quad (\mathsf{nat\text{-}eq\text{-}dec} \cdot r \cdot b) \cdot \\
\qquad\qquad\qquad\quad (\boldsymbol{\lambda} H' : r = b. \\
\qquad\qquad\qquad\qquad \mathsf{exI} \cdot (\lambda r.\ \exists q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b) \cdot 0 \cdot \\
\qquad\qquad\qquad\qquad\quad (\mathsf{exI} \cdot (\lambda q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + 0 \wedge 0 \leq b) \cdot \mathsf{Suc\ q} \cdot \\
\qquad\qquad\qquad\qquad\qquad (\mathsf{conjI} \cdot \mathsf{Suc}\ n = \mathsf{Suc}\ b * \mathsf{Suc}\ q + 0 \cdot 0 \leq b \cdot \ldots \cdot \ldots))) \cdot \\
\qquad\qquad\qquad\quad (\boldsymbol{\lambda} H' : r \neq b. \\
\qquad\qquad\qquad\qquad \mathsf{exI} \cdot (\lambda r.\ \exists q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b) \cdot \mathsf{Suc}\ r \cdot \\
\qquad\qquad\qquad\qquad\quad (\mathsf{exI} \cdot (\lambda q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + \mathsf{Suc}\ r \wedge \mathsf{Suc}\ r \leq b) \cdot q \cdot \\
\qquad\qquad\qquad\qquad\qquad (\mathsf{conjI} \cdot \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + \mathsf{Suc}\ r \cdot \mathsf{Suc}\ r \leq b \cdot \ldots \cdot \ldots))))))
\end{array}
$$

From proofs to programs — (3) Extracted program

```
nat-rec
  (0,
    0)

  (λn H :: nat × nat.
    case H of
      (r, q) ⇒


            case
              nat-eq-dec r b of
                Left ⇒
                 (0,
                   Suc q)

              | Right ⇒
                 (Suc r,
                   q))
```

Extraction in ML

```
fun extr d defs vs ts Ts hs (PBound i) = (defs, Bound i)

  | extr d defs vs ts Ts hs (Abst (s, SOME T, prf)) =
      let val (defs', t) = extr d defs vs []
        (T :: Ts) (dummyt :: hs) (incr_pboundvars 1 0 prf)
      in (defs', Abs (s, T, t)) end

  | extr d defs vs ts Ts hs (AbsP (s, SOME t, prf)) =
      let
        val T = etype_of thy' vs Ts t;
        val (defs', t) = extr d defs vs [] (T :: Ts) (t :: hs)
          (incr_pboundvars 0 1 prf)
      in (defs',
        if T = nullT then subst_bound (nullt, t) else Abs (s, T, t))
      end
```

Extraction in ML – continued

```
| extr d defs vs ts Ts hs (prf % SOME t) =
    let val (defs', u) = extr d defs vs (t :: ts) Ts hs prf
    in (defs', u $ t) end

| extr d defs vs ts Ts hs (prf1 %% prf2) =
    let
      val (defs', f) = extr d defs vs [] Ts hs prf1;
      val prop = Reconstruct.prop_of' hs prf2;
      val T = etype_of thy' vs Ts prop
    in
      if T = nullT then (defs', f) else
        let val (defs'', t) = extr d defs' vs [] Ts hs prf2
        in (defs'', f $ t) end
    end
```

Computational content of inductive datatypes and predicates

Proof	Program
Induction on datatype $nat\text{-}induct : P\ 0 \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow P\ (Suc\ x)) \Longrightarrow P\ z$	Recursion on datatype $nat\text{-}rec : nat \Rightarrow \alpha_P \Rightarrow (nat \Rightarrow \alpha_P \Rightarrow \alpha_P) \Rightarrow \alpha_P$
Inductive predicate Introduction rules **inductive** bar $bar1 : \bigwedge ws.\ ws \in good \Longrightarrow ws \in bar$ $bar2 : \bigwedge ws.\ (\bigwedge w.\ w\ \#\ ws \in bar) \Longrightarrow ws \in bar$	Inductive datatype Constructors **datatype** $barT =$ $bar1\ (letter\ list\ list)$ $\mid bar2\ (letter\ list\ list)\ (letter\ list \Rightarrow barT)$
Induction on derivation $bar\text{-}induct : vs \in bar \Longrightarrow (\bigwedge ws.\ ws \in good \Longrightarrow P\ ws) \Longrightarrow (\bigwedge ws.\ (\bigwedge w.\ w\ \#\ ws \in bar) \Longrightarrow (\bigwedge w.\ P\ (w\ \#\ ws)) \Longrightarrow P\ ws) \Longrightarrow P\ vs$	Recursion on datatype $bar\text{-}rec : barT \Rightarrow (letter\ list\ list \Rightarrow \alpha_P) \Rightarrow (letter\ list\ list \Rightarrow (letter\ list \Rightarrow barT) \Rightarrow (letter\ list \Rightarrow \alpha_P) \Rightarrow \alpha_P) \Rightarrow \alpha_P$

Equality in Isabelle/Pure

- Internal equality: all proofs are modulo β and η (but not modulo δ or ι)
- Axiomatized meta equality $\equiv$ [Paulson 1989]

$$\frac{f \equiv g \quad x \equiv y}{f\ x \equiv g\ y}\ \text{combination} \qquad \frac{\bigwedge x.\ f\ x \equiv g\ x}{\lambda x.\ f\ x \equiv \lambda x.\ g\ x}\ \text{abstract}$$

$$\frac{x \equiv y \quad y \equiv z}{x \equiv z}\ \text{transitive} \qquad \frac{x \equiv y}{y \equiv x}\ \text{symmetric} \qquad \frac{}{x \equiv x}\ \text{reflexive}$$

$$\frac{P \Longrightarrow Q \quad Q \Longrightarrow P}{P \equiv Q}\ \text{equal\_intr} \qquad \frac{P \equiv Q \quad P}{Q}\ \text{equal\_elim}$$

- Strange "feature" of $\equiv$: rewriting also possible on prop
- Powerful automatic procedure for (contextual) rewriting using $\equiv$
- Note: set of rewrite rules may be non-confluent and non-terminating!

Rewriting on prop

Can construct equality proofs unsuitable for normalization:

equal_elim $(A \Longrightarrow C)$ $(B \Longrightarrow D)$
  (combination $(\Longrightarrow\ A)$ $(\Longrightarrow\ B)$ C D
    (combination $\Longrightarrow$ $\Longrightarrow$ A B (reflexive $\Longrightarrow$) prf_1) prf_2) prf_3 prf_4

Better:

equal_elim $(A \Longrightarrow C)$ $(B \Longrightarrow D)$
  (equal_intr $(A \Longrightarrow C)$ $(B \Longrightarrow D)$
    $(\lambda H_1 : (A \Longrightarrow C)\ H_2 : B.$ equal_elim C D prf_2
      $(H_1$ (equal_elim B A (symmetric A B prf_1) H_2)))
    $(\lambda H_1 : (B \Longrightarrow D)\ H_2 : A.$ equal_elim D C (symmetric C D prf_2)
      $(H_1$ (equal_elim A B prf_1 H_2)))) prf_3 prf_4

Reduction rules:

equal_elim _ _ (equal_intr _ _ prf_1 prf_2) prf_3 $\mapsto$ prf_1 prf_3

equal_elim _ _ (symmetric _ _ (equal_intr _ _ prf_1 prf_2)) prf_3 $\mapsto$ prf_2 prf_3

Conclusions and future work

- Proof terms for Isabelle/Pure (simply-typed minimal higher order logic)
- Available for all object logics
- Quite effective compression: over 90% of terms can be omitted

Future work

- More compact representation for equality proofs
- Support for type class reasoning
- Independent checker (e.g. Twelf, CC)