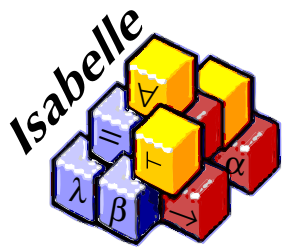
Program extraction in Isabelle

Stefan Berghofer

Institut für Informatik
TU München



Background

- Isabelle: Generic theorem prover ("logical framework")
  Developed by L. Paulson and T. Nipkow since 1986

- Primitive proof objects (TPHOLs 2000)
  Intended application: proof carrying code

- Code generation from higher order logic specification (TYPES 2000)
  Main application: rapid prototyping of programming language semantics (Java, EU VerifiCard project)

- Combining proof objects and code generation: Program extraction

Why program extraction?

- Traditional verification:

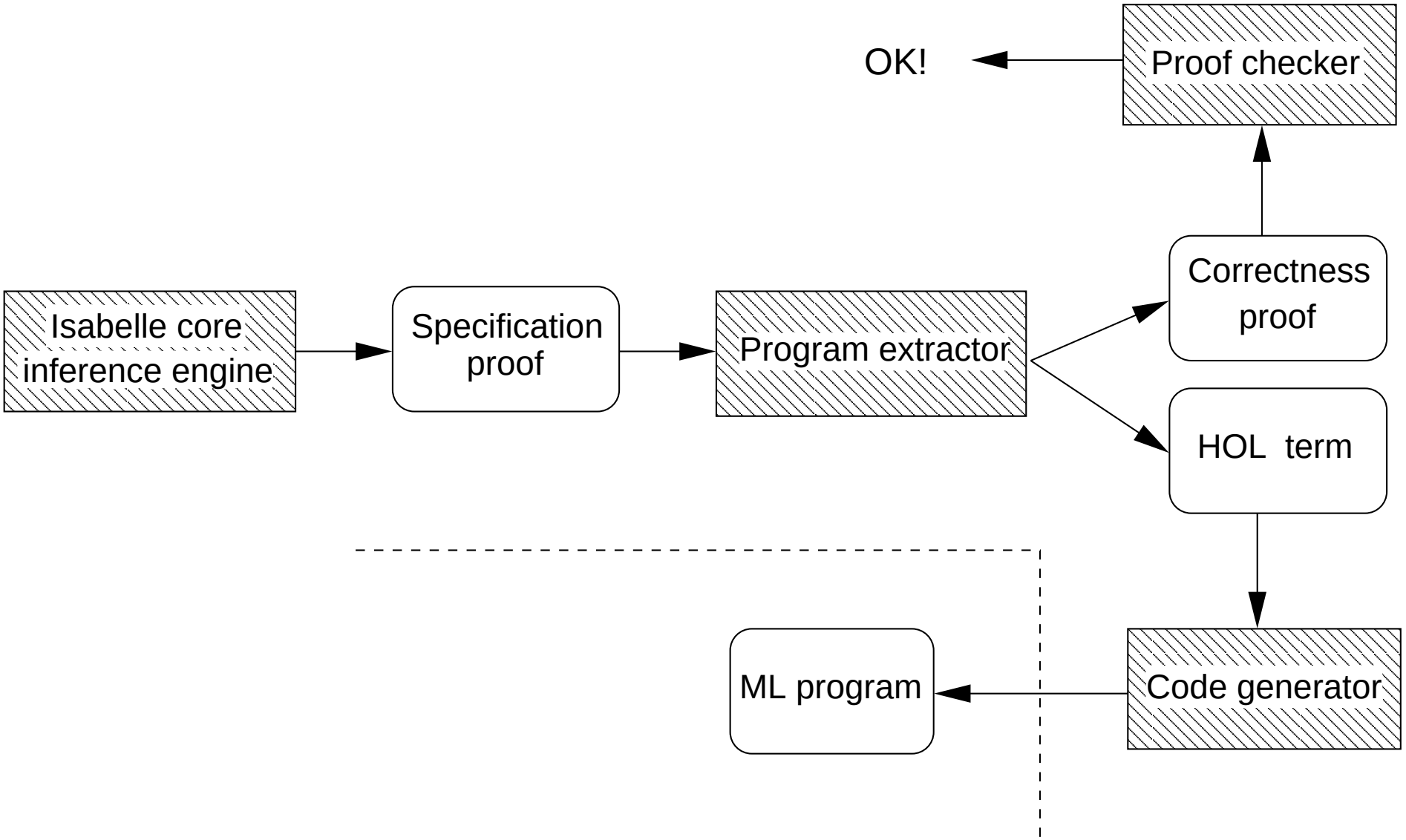
  1. write specification (logical formula)
  2. write program
  3. prove that program satisfies specification

- Problem: find proof principles (e.g. induction ...) that fit structure of program

- Program extraction:

  – systematic way of building programs that are correct by construction
  – Idea: (constructive) proof can be viewed as program + correctness proof

Program extraction infrastructure



Isabelle/HOL

- Theorem prover used for verification of functional programs (and many other things ...)
- Logical operators: $\longrightarrow$, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, ...
- Inductive datatypes

  $\mathbf{datatype}$ nat $=$ 0 | Suc nat

  $\mathbf{datatype}$ $\alpha + \beta$ $=$ Inl α | Inr β

- Recursive functions / pattern matching

  $$\begin{array}{lcl} \text{nat\_rec } f\ g\ 0 & = & f \\ \text{nat\_rec } f\ g\ (\text{Suc } n) & = & g\ n\ (\text{nat\_rec } f\ g\ n) \end{array}$$

  case t of Inl $x \Rightarrow \ldots$ | Inr $y \Rightarrow \ldots$

What is a proof?

Curry-Howard isomorphism: proofs are programs

- A proof of $A \longrightarrow B$ is a program that transforms a proof of A into a proof of B
- A proof of $\forall x.\ P\ x$ is a program that transforms an element x into a proof of $P\ x$
- A proof of $\exists x.\ P\ x$ is a pair consisting of an element x and a proof of $P\ x$
- A proof of $A \wedge B$ is a pair consisting of a proof of A and a proof of B
- A proof of $A \vee B$ is either a proof of A or a proof of B
- ...

A proof of

$$\forall a\ b.\ \exists r\ q.\ a = \mathsf{Suc}\ b * q + r \wedge r \leq b$$

is a function that takes two natural numbers a and b and returns a pair consisting of two natural numbers r and q, together with a proof of $a = \mathsf{Suc}\ b * q + r \wedge r \leq b$

The Isabelle logical framework

$$\tau, \sigma \;=\; \alpha \mid (\tau_1, \ldots, \tau_n)tc \qquad\qquad \text{where } tc \in \{\mathsf{prop}, \Rightarrow, \ldots\}$$

TYPES

$$t, u, P, Q \;=\; x \mid c_\tau \mid t\ t' \mid \lambda x :: \tau.\ t \qquad \text{where } c \in \{\textstyle\bigwedge_{(\tau \Rightarrow \mathsf{prop}) \Rightarrow \mathsf{prop}}, \Longrightarrow_{\mathsf{prop} \Rightarrow \mathsf{prop} \Rightarrow \mathsf{prop}}, \ldots\}$$

$$\frac{}{\Gamma, x :: \tau, \Gamma' \vdash x :: \tau} \qquad \frac{}{\Gamma \vdash c_\tau :: \tau} \qquad \frac{\Gamma \vdash t :: \tau \Rightarrow \sigma \quad \Gamma \vdash u :: \tau}{\Gamma \vdash t\ u :: \sigma} \qquad \frac{\Gamma, x :: \tau \vdash t :: \sigma}{\Gamma \vdash \lambda x :: \tau.\ t :: \tau \Rightarrow \sigma}$$

TERMS

$$p, q \;=\; h \mid c_{[\overline{\tau_n}/\overline{\alpha_n}]} \mid p \cdot t \mid p \centerdot q \mid \boldsymbol{\lambda} x :: \tau.\ p \mid \boldsymbol{\lambda} h : P.\ p$$

$$\frac{}{\Gamma, h : t, \Gamma' \vdash h : t} \qquad \frac{}{\Gamma \vdash c_{[\overline{\tau_n}/\overline{\alpha_n}]} : \Sigma(c)[\overline{\tau_n}/\overline{\alpha_n}]}$$

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

$$\frac{\Gamma \vdash p : P \Longrightarrow Q \quad \Gamma \vdash q : P}{\Gamma \vdash p \centerdot q : Q} \qquad \frac{\Gamma, h : P \vdash p : Q}{\Gamma \vdash \boldsymbol{\lambda} h : P.\ p : P \Longrightarrow Q}$$

PROOFS

Formalizing object logics

Textbook notation	Isabelle notation
$\dfrac{P_1 \quad \ldots \quad P_n}{Q}$	$\bigwedge P_1 \ \ldots \ P_n \ Q. \qquad P_1 \Longrightarrow \ldots \Longrightarrow P_n \Longrightarrow Q$
$\dfrac{\begin{array}{c}[P] \\ \vdots \\ Q\end{array}}{P \longrightarrow Q}$	$\bigwedge P \ Q. \qquad (P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$
$\dfrac{\begin{array}{c}[x] \\ \vdots \\ P\ x\end{array}}{\forall x.\ P\ x}$	$\bigwedge P. \qquad (\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$

Inference rules of Isabelle/HOL

impI	$\bigwedge P\ Q.$	$(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$
mp	$\bigwedge P\ Q.$	$P \longrightarrow Q \Longrightarrow P \Longrightarrow Q$
allI	$\bigwedge P.$	$(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$
spec	$\bigwedge P\ x.$	$\forall x.\ P\ x \Longrightarrow P\ x$
exI	$\bigwedge P\ x.$	$P\ x \Longrightarrow \exists x.\ P\ x$
exE	$\bigwedge P\ Q.$	$\exists x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q) \Longrightarrow Q$
conjI	$\bigwedge P\ Q.$	$P \Longrightarrow Q \Longrightarrow P \wedge Q$
conjunct_1	$\bigwedge P\ Q.$	$P \wedge Q \Longrightarrow P$
conjunct_2	$\bigwedge P\ Q.$	$P \wedge Q \Longrightarrow Q$
disjI_1	$\bigwedge P\ Q.$	$P \Longrightarrow P \vee Q$
disjI_2	$\bigwedge Q\ P.$	$Q \Longrightarrow P \vee Q$
disjE	$\bigwedge P\ Q\ R.$	$P \vee Q \Longrightarrow (P \Longrightarrow R) \Longrightarrow (Q \Longrightarrow R) \Longrightarrow R$
FalseE	$\bigwedge P.$	$\text{False} \Longrightarrow P$
nat.induct	$\bigwedge P\ n.$	$P\ 0 \Longrightarrow (\bigwedge n.\ P\ n \Longrightarrow P\ (\text{Suc}\ n)) \Longrightarrow P\ n$

Representation of proofs

Proof of $A \wedge B \longrightarrow B \wedge A$

$$\mathsf{impl} \cdot A \wedge B \cdot B \wedge A \cdot (\boldsymbol{\lambda} H : A \wedge B.\ \mathsf{conjI} \cdot B \cdot A \cdot (\mathsf{conjunct}_2 \cdot A \cdot B \cdot H) \cdot (\mathsf{conjunct}_1 \cdot A \cdot B \cdot H))$$

Proof of $A \vee B \longrightarrow B \vee A$

$$\mathsf{impl} \cdot A \vee B \cdot B \vee A \cdot (\boldsymbol{\lambda} H : A \vee B.\ \mathsf{disjE} \cdot A \cdot B \cdot B \vee A \cdot H \cdot (\mathsf{disjI}_2 \cdot A \cdot B) \cdot (\mathsf{disjI}_1 \cdot B \cdot A))$$

Proof of $(\exists x.\ \forall y.\ P\ x\ y) \longrightarrow (\forall y.\ \exists x.\ P\ x\ y)$

$$\mathsf{impl} \cdot \exists x.\ \forall y.\ P\ x\ y \cdot \forall y.\ \exists x.\ P\ x\ y \cdot (\boldsymbol{\lambda} H : \exists x.\ \forall y.\ P\ x\ y.\ \mathsf{allI} \cdot (\lambda y.\ \exists x.\ P\ x\ y) \cdot (\boldsymbol{\lambda} y.\ \mathsf{exE} \cdot (\lambda x.\ \forall y.\ P\ x\ y) \cdot \exists x.\ P\ x\ y \cdot H \cdot (\boldsymbol{\lambda} x\ H : \forall y.\ P\ x\ y.\ \mathsf{exI} \cdot (\lambda x.\ P\ x\ y) \cdot x \cdot (\mathsf{spec} \cdot P\ x \cdot y \cdot H))))$$

From proofs to programs

- What does it mean for an extracted program to be correct?
  ≫ Realizability interpretation: realizes $p\ P$

- How to extract the program?
  ≫ Map inference rules r in proof to programs $\mathcal{E}(r)$

- How to obtain the correctness proof?
  ≫ Map inference rules r in proof to correctness proofs $\mathcal{C}(r)$
  ≫ For $\vdash r : P$, show $\vdash \mathcal{C}(r)$: realizes $\mathcal{E}(r)\ P$

- Not all parts of proof perform computations, e.g.

$$\overbrace{\forall a\ b.\ \exists r\ q.}^{\text{computation}}\ \underbrace{a = \mathsf{Suc}\ b * q + r \wedge r \leq b}_{\text{verification of result}}$$

  ≫ programs extracted from computationally irrelevant subproofs have type unit

Realizability

Establishes a relationship between programs and specifications

$$\begin{array}{lcl}
\text{realizes } (t :: \alpha \Rightarrow \beta)\ (A \Longrightarrow B) & \equiv & \bigwedge x.\ \text{realizes } x\ A \Longrightarrow \text{realizes } (t\ x)\ B \\
\text{realizes } (t :: \alpha \Rightarrow \beta)\ (\bigwedge x.\ P\ x) & \equiv & \bigwedge x.\ \text{realizes } (t\ x)\ (P\ x) \\
 & & \\
\text{realizes } (t :: \text{unit})\ P & \equiv & P \\
\text{realizes } (t :: \alpha \Rightarrow \beta)\ (A \longrightarrow B) & \equiv & \forall x.\ \text{realizes } x\ A \longrightarrow \text{realizes } (t\ x)\ B \\
\text{realizes } (t :: \alpha \Rightarrow \beta)\ (\forall x.\ P\ x) & \equiv & \forall x.\ \text{realizes } (t\ x)\ (P\ x) \\
\text{realizes } (t :: \alpha \times \beta)\ (\exists x.\ P\ x) & \equiv & \text{realizes } (\text{snd } t)\ (P\ (\text{fst } t)) \\
\text{realizes } (t :: \alpha + \beta)\ (P \vee Q) & \equiv & \text{case } t \text{ of Inl } p \Rightarrow \text{realizes } p\ P \mid \text{Inr } q \Rightarrow \text{realizes } q\ Q \\
\text{realizes } (t :: \alpha \times \beta)\ (P \wedge Q) & \equiv & \text{realizes } (\text{fst } t)\ P \wedge \text{realizes } (\text{snd } t)\ Q \\
\text{realizes } t\ \text{False} & \equiv & \text{False}
\end{array}$$

Example

$$\text{realizes division } (\forall a\ b.\ \exists r\ q.\ a = \text{Suc } b * q + r \wedge r \leq b) \equiv$$
$$\forall a\ b.\ a = \text{Suc } b * \text{fst } (\text{snd } (\text{division } a\ b)) + \text{fst } (\text{division } a\ b) \wedge \text{fst } (\text{division } a\ b) \leq b$$

where division :: nat $\Rightarrow$ nat $\Rightarrow$ nat $\times$ (nat $\times$ (unit $\times$ unit))

Extracting the program

How to find out type of extracted program?

- From formula
  Problem: need many different versions of logical operators (e.g. in Coq):

```
Inductive ex [A:Set;P:A->Prop] : Prop
  := ex_intro : (x:A)(P x)->(ex A P).

Inductive sig [A:Set;P:A->Prop] : Set
  := exist : (x:A)(P x) -> (sig A P).

Inductive sigS [A:Set;P:A->Set] : Set
  := existS : (x:A)(P x) -> (sigS A P).
```

- From proof: Why not just use type inference?

$$\text{extr}\ \underbrace{\Gamma}_{\text{variable context}}\ \overbrace{\theta}^{\text{type variable assignment}}\ \underbrace{p}_{\text{proof}} = (\ \overbrace{t}^{\text{term}}\ ,\ \underbrace{\tau}_{\text{type}}\ ,\ \overbrace{\theta'}^{\text{new type variable assignment}}\ )$$

Extracting the program — continued

$$\begin{array}{lcl}
\mathsf{extr}\ \Gamma\ \theta\ c & = & (t[\overline{?\beta_n}/\overline{\alpha_n}],\ \tau[\overline{?\beta_n}/\overline{\alpha_n}],\ \theta) \\
 & & \quad \text{where} \quad t = \mathcal{E}(c) \quad \Gamma \vdash t :: \tau \quad \overline{\alpha_n} = \mathsf{FV}(\tau) \\
\mathsf{extr}\ \Gamma\ \theta\ h & = & (\underline{h},\ \Gamma(\underline{h}),\ \theta) \\
\mathsf{extr}\ \Gamma\ \theta\ (\boldsymbol{\lambda} x :: \tau.\ p) & = & (\lambda x :: \tau.\ t,\ \tau \Rightarrow \sigma,\ \theta') \\
 & & \quad \text{where } (t,\ \sigma,\ \theta') = \mathsf{extr}\ (\Gamma, x :: \tau)\ \theta\ p \\
\mathsf{extr}\ \Gamma\ \theta\ (\boldsymbol{\lambda} h : P.\ p) & = & (\lambda \underline{h} :: ?\alpha.\ t,\ ?\alpha \Rightarrow \sigma,\ \theta') \\
 & & \quad \text{where } (t,\ \sigma,\ \theta') = \mathsf{extr}\ (\Gamma, \underline{h} :: ?\alpha)\ \theta\ p \\
\mathsf{extr}\ \Gamma\ \theta\ (p \cdot t) & = & \begin{cases} ((),\ \mathsf{unit},\ \theta) & \text{if } \theta_2 = \bot \\ (f\ t,\ ?\alpha,\ \theta_2 \circ \theta_1) & \text{otherwise} \end{cases} \\
 & & \quad \text{where} \quad (f,\ \tau,\ \theta_1) = \mathsf{extr}\ \Gamma\ \theta\ p \quad \Gamma \vdash t :: \sigma \\
 & & \qquad\quad \theta_2 = \mathsf{mgu}\ (\theta_1(\tau))\ (\theta_1(\sigma \Rightarrow ?\alpha)) \\
\mathsf{extr}\ \Gamma\ \theta\ (p \cdot q) & = & \begin{cases} (f\ (),\ \sigma',\ \theta_1) & \text{if } \theta_1(\tau) = \mathsf{unit} \Rightarrow \sigma' \\ ((),\ \mathsf{unit},\ \theta) & \text{if } \theta_3 = \bot \\ (f\ t,\ ?\alpha,\ \theta_3 \circ \theta_2) & \text{otherwise} \end{cases} \\
 & & \quad \text{where} \quad (f,\ \tau,\ \theta_1) = \mathsf{extr}\ \Gamma\ \theta\ p \\
 & & \qquad\quad (t,\ \sigma,\ \theta_2) = \mathsf{extr}\ \Gamma\ \theta_1\ q \\
 & & \qquad\quad \theta_3 = \mathsf{mgu}\ (\theta_2(\tau))\ (\theta_2(\sigma \Rightarrow ?\alpha))
\end{array}$$

Extracting the program — type inference

$p \;=\; \mathsf{exE} \cdot P \cdot Q$
$\mathsf{extr}\ \Gamma\ \theta\ p \;=\; (f,\ ?\alpha \times ?\beta \Rightarrow (?\alpha \Rightarrow ?\beta \Rightarrow ?\gamma) \Rightarrow ?\gamma,\ \theta_1)$

$\Gamma \vdash q : \exists x.\ P\ x$

- q informative

$$
\begin{array}{ll}
 & q \;=\; \mathsf{exI} \cdot \ldots \cdot \ldots \cdot \ldots \\
\rightsquigarrow & \mathsf{extr}\ \Gamma\ \theta_1\ q \;=\; (t,\ \tau_1 \times \tau_2,\ \theta_2) \\
\rightsquigarrow & \mathsf{extr}\ \Gamma\ \theta\ (p \cdot q) \;=\; (f\ t,\ (\tau_1 \Rightarrow \tau_2 \Rightarrow ?\gamma) \Rightarrow ?\gamma,\ \{?\alpha \mapsto \tau_1,\ ?\beta \mapsto \tau_2,\ \ldots\})
\end{array}
$$

- q non-informative

$$
\begin{array}{ll}
 & q \;=\; \mathsf{classical} \cdot \ldots \cdot \ldots \\
\rightsquigarrow & \mathsf{extr}\ \Gamma\ \theta_1\ q \;=\; ((),\ \mathsf{unit},\ \theta_2) \\
\rightsquigarrow & \mathsf{extr}\ \Gamma\ \theta\ (p \cdot q) \;=\; ((),\ \mathsf{unit},\ \theta)
\end{array}
$$

Producing the correctness proof

$$\mathsf{corr}\ \Gamma\ p\ t = \begin{cases} p & \text{if } \Gamma \vdash t :: \mathsf{unit} \\ \mathsf{corr}'\ \Gamma\ p\ t & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\mathsf{corr}'\ \Gamma\ c\ _ &= \mathcal{C}(c) \\
\mathsf{corr}'\ \Gamma\ h\ _ &= h \\
\mathsf{corr}'\ \Gamma\ (\boldsymbol{\lambda} x :: _.\ p)\ (\lambda x :: \tau.\ t) &= \boldsymbol{\lambda} x :: \tau.\ \mathsf{corr}\ (\Gamma, x :: \tau)\ p\ t \\
\mathsf{corr}'\ \Gamma\ (\boldsymbol{\lambda} h : P.\ p)\ (\lambda \underline{h} :: \tau.\ t) &= \boldsymbol{\lambda}(\underline{h} :: \tau)\ h : \mathsf{realizes}\ \underline{h}\ P.\ \mathsf{corr}\ (\Gamma, \underline{h} :: \tau)\ p\ t \\
\mathsf{corr}'\ \Gamma\ (p \cdot _)\ (f\ t) &= \mathsf{corr}\ \Gamma\ p\ u \cdot t \\
\mathsf{corr}'\ \Gamma\ (p \bullet q)\ (f\ t) &= \mathsf{corr}\ \Gamma\ p\ f \cdot t \bullet \mathsf{corr}\ \Gamma\ q\ t
\end{aligned}$$

Theorem

let $(t, \ \ldots, \ \theta') = \mathsf{extr}\ \Gamma\ \theta\ p$

and $\vdash p : \varphi$

then $\vdash \mathsf{corr}\ \Gamma\ p\ (\theta'(t)) : \mathsf{realizes}\ (\theta'(t))\ \varphi$

Proof: Induction on p

Programs corresponding to inference rules

impI	$\lambda P\ Q\ (pq :: \alpha \Rightarrow \beta).\ pq$
mp	$\lambda P\ Q\ (pq :: \alpha \Rightarrow \beta).\ pq$
allI	$\lambda P\ (p :: \alpha \Rightarrow \beta).\ p$
spec	$\lambda P\ x\ p.\ p\ x$
exI	$\lambda P\ x\ p.\ (x,\ p)$
exE	$\lambda P\ Q\ p\ pq.\ pq\ (\mathsf{fst}\ p)\ (\mathsf{snd}\ p)$
conjI	$\lambda P\ Q\ p\ q.\ (p,\ q)$
$\mathsf{conjunct}_1$	$\lambda P\ Q.\ \mathsf{fst}$
$\mathsf{conjunct}_2$	$\lambda P\ Q.\ \mathsf{snd}$
disjI_1	$\lambda P\ Q.\ \mathsf{Inl}$
disjI_2	$\lambda P\ Q.\ \mathsf{Inr}$
disjE	$\lambda P\ Q\ R\ pq\ pr\ qr.\ (\mathsf{case}\ pq\ \mathsf{of}\ \mathsf{Inl}\ p \Rightarrow pr\ p \mid \mathsf{Inr}\ q \Rightarrow qr\ q)$
FalseE	$\lambda P\ (p :: \mathsf{unit}).\ \mathsf{arbitrary}$
nat.induct	$\lambda P\ n\ p_0\ p_S.\ \mathsf{nat\_rec}\ p_0\ p_S\ n$

Correctness of programs corresponding to inference rules (1)

$$\begin{aligned}
\mathcal{C}(\mathsf{impl}) \quad = \quad & \boldsymbol{\lambda} P\ Q\ pq\ (h : \bigwedge u.\ \mathsf{realizes}\ u\ P \Longrightarrow \mathsf{realizes}\ (pq\ u)\ Q). \\
& \quad \mathsf{alll} \cdot (\lambda x.\ \mathsf{realizes}\ x\ P \longrightarrow \mathsf{realizes}\ (pq\ x)\ Q) \cdot \\
& \quad\ (\boldsymbol{\lambda} x.\ \mathsf{impl} \cdot \mathsf{realizes}\ x\ P \cdot \mathsf{realizes}\ (pq\ x)\ Q \cdot (h \cdot x))
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}(\mathsf{exE}) \quad = \quad & \boldsymbol{\lambda} P\ Q\ p\ (h_1 : \mathsf{realizes}\ (\mathsf{snd}\ p)\ (P\ (\mathsf{fst}\ p)))\ pq \\
& \quad (h_2 : \bigwedge x\ x'.\mathsf{realizes}\ x'\ (P\ x) \Longrightarrow \mathsf{realizes}\ (pq\ x\ x')\ Q). \\
& \quad\ h_2 \cdot \mathsf{fst}\ p \cdot \mathsf{snd}\ p \cdot h_1
\end{aligned}$$

Correctness of programs corresponding to inference rules (2)

What to do with higher order variables?

$$\text{realizes } t\ (P\ \overline{t_n}) \quad \equiv \quad P'\ t\ \overline{t_n}$$

Example: induction on natural numbers

$$\text{realizes } (\lambda p_0\ p_S.\ \text{nat\_rec}\ p_0\ p_S\ n)\ (P\ 0 \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow P\ (\text{Suc}\ x)) \Longrightarrow P\ n) \equiv$$

$$\bigwedge p_0.\ \text{realizes}\ p_0\ (P\ 0) \Longrightarrow (\bigwedge p_S.\ \text{realizes}\ p_S\ (\bigwedge x.\ P\ x \Longrightarrow P\ (\text{Suc}\ x)) \Longrightarrow \text{realizes}\ (\text{nat\_rec}\ p_0\ p_S\ n)\ (P\ n)) \equiv$$

$$\bigwedge p_0.\ \text{realizes}\ p_0\ (P\ 0) \Longrightarrow (\bigwedge p_S.\ (\bigwedge x.\ \text{realizes}\ (p_S\ x)\ (P\ x \Longrightarrow P\ (\text{Suc}\ x))) \Longrightarrow \text{realizes}\ (\text{nat\_rec}\ p_0\ p_S\ n)\ (P\ n)) \equiv$$

$$\bigwedge p_0.\ \text{realizes}\ p_0\ (P\ 0) \Longrightarrow (\bigwedge p_S.\ (\bigwedge x\ h.\ \text{realizes}\ h\ (P\ x) \Longrightarrow \text{realizes}\ (p_S\ x\ h)\ (P\ (\text{Suc}\ x))) \Longrightarrow \text{realizes}\ (\text{nat\_rec}\ p_0\ p_S\ n)\ (P\ n)) \equiv$$

$$\underbrace{\bigwedge p_0.\ P'\ p_0\ 0 \Longrightarrow (\bigwedge p_S.\ (\bigwedge x\ h.\ P'\ h\ x \Longrightarrow P'\ (p_S\ x\ h)\ (\text{Suc}\ x)) \Longrightarrow P'\ (\text{nat\_rec}\ p_0\ p_S\ n)\ n)}_{\varphi}$$

Proof: Induction on n

$$\mathcal{C}(\text{nat\_ind}) \ = \ \boldsymbol{\lambda} P\ n.\ \text{nat\_ind\_realizer} \cdot (\lambda t\ x.\ \text{realizes}\ t\ (P\ x)) \cdot n$$

where $\quad \vdash \text{nat\_ind\_realizer} : \bigwedge P'\ n.\ \varphi$

Synthesizing a division algorithm (1)

$$
\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 = b \cdot \exists r\ q.\ \mathsf{Suc}\ n = \mathsf{Suc}\ b * q + r \wedge r \leq b \cdot \\
\qquad\qquad\qquad\quad (\mathsf{le\_lt\_eq\_dec} \cdot r \cdot b \cdot (\mathsf{conjunct}_2 \cdot n = \mathsf{Suc}\ b * q + r \cdot r \leq b \cdot H)) \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 \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))) \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)))))))
\end{array}
$$

Synthesizing a division algorithm (2)

```
nat_rec
  (0,
    (0,
      ((), ())))
  (λn H :: nat × (nat × (unit × unit)).
     case
       le_lt_eq_dec (fst H) b () of
         Inl (H' :: unit) ⇒
          (Suc (fst H),
            (fst (snd H),
              ((), ())))
       | Inr (H' :: unit) ⇒
          (0,
            (Suc (fst (snd H)),
              ((), ())))) a
```

Program extraction for inductive predicates

inductive rtrancl r

rtrancl_refl : $(a, a) \in$ rtrancl r

rtrancl_into_rtrancl : $(a, b) \in$ rtrancl $r \Longrightarrow (b, c) \in r \Longrightarrow (a, c) \in$ rtrancl r

Realizers for introduction rules

datatype (α, β)rtr $=$

Refl α

| Into α α α $((\alpha, \beta)\text{rtr})$ β

consts

rtr_realizes :: $(\beta \Rightarrow (\alpha \times \alpha) \Rightarrow \text{bool}) \Rightarrow (\alpha, \beta)\text{rtr} \Rightarrow (\alpha \times \alpha) \Rightarrow \text{bool}$

primrec

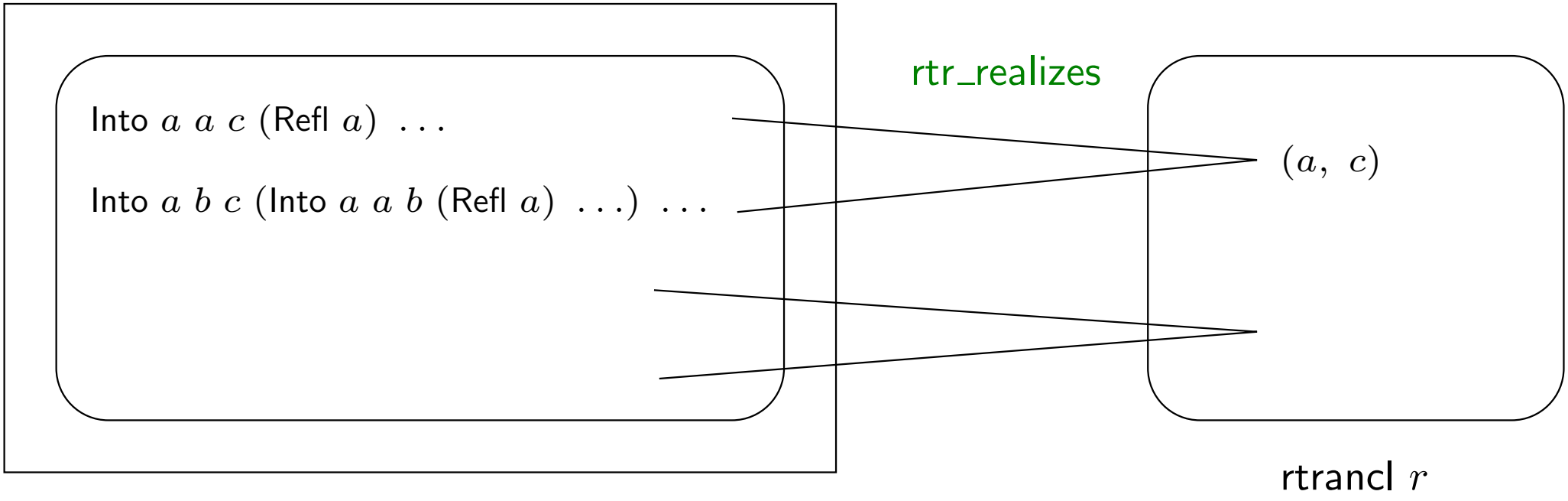
rtr_realizes R (Refl a) aa $=$ (fst $aa = a \wedge$ snd $aa = a$)

rtr_realizes R (Into a b c p_{ab} p_{bc}) ac $=$

(fst $ac = a \wedge$ snd $ac = c \wedge$ rtr_realizes R p_{ab} $(a, b) \wedge R$ p_{bc} (b, c))

realizes $(t :: (\alpha, \beta)\text{rtr})$ $(xy \in \text{rtrancl } r) \equiv$ rtr_realizes $(\lambda p_{bc}$ bc. realizes p_{bc} $(bc \in r))$ t xy

Realizability for inductive predicates



$$r \quad = \quad \{(a,\ b),\ (b,\ c),\ (a,\ c),\ \ldots\}$$

Inductive predicates — realizer for induction rule

rtrancl_induct : $(x, y) \in \text{rtrancl } r \Longrightarrow (\bigwedge a.\ P\ a\ a) \Longrightarrow$
$(\bigwedge a\ b\ c.\ (a, b) \in \text{rtrancl } r \Longrightarrow P\ a\ b \Longrightarrow (b, c) \in r \Longrightarrow P\ a\ c) \Longrightarrow P\ x\ y$

consts

rtr_ind :: $(\alpha, \beta)\text{rtr} \Rightarrow (\alpha \Rightarrow \gamma) \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha \Rightarrow (\alpha, \beta)\text{rtr} \Rightarrow \gamma \Rightarrow \beta \Rightarrow \gamma) \Rightarrow \gamma$

primrec

rtr_ind (Refl a) $f\ g = f\ a$
rtr_ind (Into $a\ b\ c\ ab\ bc$) $f\ g = g\ a\ b\ c\ ab$ (rtr_ind $ab\ f\ g$) bc

Correctness theorem

$\bigwedge p_{xy}.$ rtr_realizes $R\ p_{xy}\ (x, y) \Longrightarrow (\bigwedge f.\ (\bigwedge a.\ P\ (f\ a)\ a\ a) \Longrightarrow$
$(\bigwedge g.\ (\bigwedge a\ b\ c\ p_{ab}.$ rtr_realizes $R\ p_{ab}\ (a, b) \Longrightarrow (\bigwedge p.\ P\ p\ a\ b \Longrightarrow$
$(\bigwedge p_{bc}.\ R\ p_{bc}\ (b, c) \Longrightarrow P\ (g\ a\ b\ c\ p_{ab}\ p\ p_{bc})\ a\ c))) \Longrightarrow$
P (rtr_ind $p_{xy}\ f\ g)\ x\ y))$

Problems combining computationally relevant / irrelevant proofs

$\boldsymbol{\lambda} H : \varphi.$ [$\ldots\ H\ \ldots$ [$\ldots\ H\ \ldots$]]

Correctness proof: $\boldsymbol{\lambda}(x :: \tau)\ H$: realizes $x\ \varphi.$ [$\ldots\ H\ \ldots$ [$\ldots\ H\ \ldots$]]

Problem

If H computationally relevant (i.e. $\tau \neq$ unit), cannot just copy subproof [$\ldots\ H\ \ldots$] unchanged into correctness proof

Solutions

1. Insert additional subproof p: $\boldsymbol{\lambda}(x :: \tau)\ H$: realizes $x\ \varphi.$ [$\ldots\ H\ \ldots$ [$\ldots\ p\ H\ \ldots$]]

   where p : realizes $x\ \varphi \Longrightarrow \varphi$ (Note: not directly provable in intuitionistic logic)

2. Use different definition of realizability:
   q_realizes $f\ (P \longrightarrow Q) \equiv (P \longrightarrow Q) \wedge (\forall x.$ q_realizes $x\ P \longrightarrow$ q_realizes $(f\ x)\ Q)$

Future work

- Better optimization of extracted programs
  Eliminate unnecessary $()$'s

- Instantiate to other logics (e.g. ZF)
  Problem: type systems of meta logic and object logic need to coincide

- Larger case studies (e.g. graph theory)

- Program extraction from classical proofs?

Related work

- Deductive Tableaux (Manna & Waldinger)
- Theorem provers based on type theory:
  - NuPRL (Constable)
  - Coq (Coquand, Huet, Paulin-Mohring)
- Minlog (Schwichtenberg)
- Extraction of structured programs from CASL specification proofs (Wirsing, Crossley)

Realistic applications of extraction

- Type checker for Calculus of Constructions (B. Barras)
- Buchberger's algorithm (L. Théry)
- Java Card tokenisation algorithm (E. Denney)