Autumn school "Proof and Computation"

Herrsching, 14-20 September 2025

Program extraction in higher-order logic

Ulrich Berger

Swansea University

Introduction

In Constructive Mathematics,

objects proven to exist can be constructed,

proven disjunctions can be effectively decided,

and proofs of implications or universally quantified statements give rise to algorithms for computable functions.

Program extraction is the process of extracting these algorithms.

Program extraction is implemented in various proof systems, for example, NuprL, Coq (now Rocq), Minlog.

The fundamental ideas underlying program extraction are known as the *Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic*, or the *Curry-Howard correspondence*.

Program extraction in type theory

In formalizations of constructive mathematics, such as constructive type theory, proofs themselves are programs.

Therefore, in type theory, program extraction is viewed as *code optimization* or *compilation*.

The correctness of extracted programs is defined with respect to the operational semantics of type theory and depends on the implementation:

> "*However, for such executables obtained by extraction, the extraction process is part of the trusted code base (TCB), as are Coq's kernel and the compiler used to compile the extracted code. The extraction process contains intricate semantic transformation of programs that rely on subtle operational features of both the source and target language.*" (Forster, Sozeau, Tabareau, 2024)

Program extraction via realizability (Kleene 1945)

(1) Each formula A is interpreted as a predicate $\mathbf{R}(A)$ that defines the computational problem expressed by A.

(2) From a formal proof d of A one extracts:

  (a) a program $\mathbf{ep}(d)$
  (b) a formal proof that $\mathbf{ep}(d)$ satisfies $\mathbf{R}(A)$ (soundness)
  (c) a typing $\vdash \mathbf{ep}(d) : \mathcal{T}(A)$ of the extracted program

Minlog's program extraction is based on realizability (Schwichtenberg, Wainer 2012, Ch. 7).

Program extraction for higher-order logic

In this lectures we study how to apply program extraction via realizability to higher-order logic.

We work with an intuitionistic version of Church's *simple theory of types* (CST, Church 1940) which formalizes higher-order logic as an instance of the *simply typed lambda calculus* (STL).

The simply typed lambda calculus serves as a logical framework for the formalization of higher-order logic, similar to the logical framework LF (Pfenning 2013).

Plan for the lectures

Lecture 1 Program extraction in Church's Simple Theory of Types

Lecture 2 Examples:

- Continuous functions, integration
- Martin Hofmann's breadth-first search
- Non-monotone induction

Lecture 3 Avoiding garbage in extracted programs
Computational adequacy
Comparison with cAERN

References for Lecture 1

H. Barendregt.
Lambda calculi with types.
Handbook of Logic in Computer Science, vol 2, p. 117–309, 1992.

A. Church.
A formulation of the simple theory of types.
The Journal of Symbolic Logic, 5(2):56–68, 1940.

S. C. Kleene.
On the interpretation of intuitionistic number theory.
The Journal of Symbolic Logic, 10:109–124, 1945.

H. Schwichtenberg and S. S. Wainer.
Proofs and Computations.
Cambridge University Press, 2012.

F. Pfenning.
Logical frameworks.
In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 977–1061. Elsevier Science and MIT Press, 2001.

References for Lecture 1 (ctd.)

B and H. Tsuiki.
Intuitionistic fixed point logic.
Annals of Pure and Applied Logic, 172(3), 2021.

P. Letouzey.
A New Extraction for Coq.
TYPES 2002, volume 2646 of *LNCS*, pages 200–219. 2003.

B and T. Hou.
A realizability interpretation of Church's simple theory of types.
Mathematical Structures in Computer Science, 27(8):1364–1385, 2017.

Y. Forster, M. Sozeau, and N. Tabareau.
Verified Extraction from Coq to OCaml.
hal-04329663, 2023.

H. Schwichtenberg.
Minlog.
In *The Seventeen Provers of the World*. *LNAI*, p. 151–157, 2006.

References for Lecture 1 (ctd.)

B, K. Miyamoto, H. Schwichtenberg, and M. Seisenberger.
Minlog - a tool for program extraction supporting algebra and coalgebra.
In *CALCO-Tools*, volume 6859 of *LNCS*, pages 393–399. 2011.

N. P. Mendler.
Inductive types and type constraints in the second-order lambda calculus.
Annals of Pure and Applied Logic, 52:159–172, 1991.

A. Abel, R. Matthes, and T. Uustalu.
Iteration and coiteration schemes for higher-order and nested datatypes.
TCS, 333:3–66, 2005.

A. Abel, B. Pientka, and A. Setzer.
Copatterns: Programming infinite structures by observations.
In POPL'13, p. 27–38, 2013.

References for Lecture 2

K. Miyamoto, F. Forsberg, and H. Schwichtenberg.
Program Extraction from Nested Definitions.
Interactive Theorem Proving, volume 7988 of LNCS, pages 370–385. 2013.

N. Ghani, P. Hancock, and D. Pattinson.
Representations of stream processors using nested fixed points.
Logical Methods in Computer Science, 5, 2009.

B.
From coinductive proofs to exact real arithmetic.
In *Computer Science Logic*, volume 5771 of *LNCS*, pages 132–146. 2009.

H. Schwichtenberg.
Inverting monotone functions in constructive analysis.
CiE 2006, volume 3988 of *LNCS*, pages 490–504. 2006.

B and M. Seisenberger.
Proofs, programs, processes.
Theory of Computing Systems, 51(3):213–329, 2012.

References for Lecture 2 (ctd.)

M. Hofmann.
Non strictly positive datatypes in system F.
www.seas.upenn.edu/~sweirich/types/archive/1993/msg00027.html.
Types Forum, 15 February 1993.

B, R. Matthes, and A. Setzer.
Martin Hofmann's case for non-strictly positive data types.
volume 130 of *LIPIcs*, Dagstuhl, Germany, 2019.

P. Aczel.
An introduction to inductive definitions.
In *Handbook of Mathematical Logic*, volume 2, pages 739–782.
North-Holland, 1977.

W. Pohlers.
Ordinal analysis of non-monotone π^0_1-definable inductive definitions.
Annals of Pure and Applied Logic, 156:160–169, 2008.

References for Lecture 2 (ctd.)

H. Schwichtenberg, M. Seisenberger, and F. Wiesnet.
Higman's lemma and its computational content.
In *Advances in Proof Theory*. Birkhäuser, 2016.

B and H. Schwichtenberg.
The greatest common divisor: a case study for program extraction from classical proofs.
TYPES '95. Volume 1158 of *LNCS*, pages 36–46. 1996.

B, H. Schwichtenberg, and M. Seisenberger.
The Warshall algorithm and Dickson's lemma: Two examples of realistic program extraction.
Journal of Automated Reasoning, 26(2), 2001.

S. Berghofer.
Program Extraction in simply-typed Higher Order Logic.
In *TYPES'02*, volume 2646 of *LNCS*, pages 21–38. 2003.

References for Lecture 3

D. Ratiu and H. Schwichtenberg.
Decorating proofs.
In S. Feferman and W. Sieg, editors, *Proofs, Categories and Computations. Essays in honor of Grigori Mints*, pages 171–188. College Publications, 2010.

G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott.
Continuous Lattices and Domains, volume 93 of *Encyclopedia of Mathematics and its Applications*.
Cambridge University Press, 2003.

aern2-real: A Haskell library for exact real number computation.
https://hackage.haskell.org/package/aern2-real, 2021.

M. Konečný, S. Park, and H. Thies.
Extracting efficient exact real number computation from proofs in constructive type theory.
Journal of Logic and Computation, Volume 35, Issue 6, September 2025.

Lecture 1

Program extraction in Church's Simple Theory of Types

The simply typed lambda calculus (STL, Barendregt 1992)

Given a set BT of *base types*, the set of *types* of STL is defined by the grammar

$$\mathrm{TYP} \ni \rho, \sigma \quad ::= \quad b \in \mathrm{BT} \mid \rho \to \sigma \mid \rho \times \sigma$$

Given a set C of *constants* and a set VAR of *variables*, the set of *terms* of STL is defined by the grammar

$$\begin{aligned} \mathrm{TER} \ni M, N \quad ::= \quad & x \in \mathrm{VAR} \mid c \in \mathrm{C} \mid \\ & \mid \lambda x : \rho . M \mid M\, N \mid \langle M, N \rangle \mid \pi_0(M) \mid \pi_1(M) \end{aligned}$$

For every constant $c \in \mathrm{C}$ we assume a type assignment $c : \rho$.

Type assignment for STL

$$\frac{}{\Gamma, x : \rho \vdash x : \rho} \qquad \frac{}{\Gamma \vdash c : \rho}\; c : \rho$$

$$\frac{\Gamma, x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x : \rho \,.\, M : \rho \to \sigma} \qquad \frac{\Gamma \vdash M : \rho \to \sigma \qquad \Gamma \vdash N : \rho}{\Gamma \vdash M\,N : \sigma}$$

$$\frac{\Gamma \vdash M : \rho \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \langle M, N \rangle : \rho \times \sigma}$$

$$\frac{\Gamma \vdash M : \rho \times \sigma}{\Gamma \vdash \pi_0(M) : \rho} \qquad \frac{\Gamma \vdash M : \rho \times \sigma}{\Gamma \vdash \pi_1(M) : \sigma}$$

Type assignment for STL

$$\frac{}{\Gamma, x : \rho \vdash x : \rho} \qquad \frac{}{\Gamma \vdash c : \rho}\; c : \rho$$

$$\frac{\Gamma, x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x : \rho \, . \, M : \rho \to \sigma} \qquad \frac{\Gamma \vdash M : \rho \to \sigma \qquad \Gamma \vdash N : \rho}{\Gamma \vdash M\, N : \sigma}$$

$$\frac{\Gamma \vdash M : \rho \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \langle M, N \rangle : \rho \times \sigma}$$

$$\frac{\Gamma \vdash M : \rho \times \sigma}{\Gamma \vdash \pi_0(M) : \rho} \qquad \frac{\Gamma \vdash M : \rho \times \sigma}{\Gamma \vdash \pi_1(M) : \sigma}$$

STL is parametric in the set BT of base types and the set of C of typed constants $c : \rho$

Each choice of these parameters defines an *instance* of STL.

Type assignment for STL

$$\frac{}{\Gamma, x : \rho \vdash x : \rho} \qquad \frac{}{\Gamma \vdash c : \rho}\ c : \rho$$

$$\frac{\Gamma, x : \rho \vdash M : \sigma}{\Gamma \vdash \lambda x : \rho \,.\, M : \rho \to \sigma} \qquad \frac{\Gamma \vdash M : \rho \to \sigma \qquad \Gamma \vdash N : \rho}{\Gamma \vdash M\, N : \sigma}$$

$$\frac{\Gamma \vdash M : \rho \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \langle M, N \rangle : \rho \times \sigma}$$

$$\frac{\Gamma \vdash M : \rho \times \sigma}{\Gamma \vdash \pi_0(M) : \rho} \qquad \frac{\Gamma \vdash M : \rho \times \sigma}{\Gamma \vdash \pi_1(M) : \sigma}$$

STL is parametric in the set BT of base types and the set of C of typed constants $c : \rho$

Each choice of these parameters defines an *instance* of STL.

For brevity we will ignore product types.

$\beta\eta$-equality

$$(\lambda x : \rho \,.\, M)\, N =_\beta M[N/x]$$

$\lambda x : \rho \,.\, (M\, x) =_\eta M$ provided x is not free in M.

There are no extra equations for the constants.

$\beta\eta$-equality

$$(\lambda x : \rho \,.\, M)\, N =_\beta M[N/x]$$

$$\lambda x : \rho \,.\, (M\, x) =_\eta M \text{ provided } x \text{ is not free in } M.$$

There are no extra equations for the constants.

$\beta\eta$-equality is the congruence on terms generated by $=_\beta \cup =_\eta$.

Due to strong normalization and confluence of the corresponding reduction relation, $\beta\eta$-equality is a decidable relation on typeable terms.

$\beta\eta$-equality

$$(\lambda x : \rho \,.\, M)\, N =_\beta M[N/x]$$

$$\lambda x : \rho \,.\, (M\, x) =_\eta M \text{ provided } x \text{ is not free in } M.$$

There are no extra equations for the constants.

$\beta\eta$-equality is the congruence on terms generated by $=_\beta \cup =_\eta$.

Due to strong normalization and confluence of the corresponding reduction relation, $\beta\eta$-equality is a decidable relation on typeable terms.

We will identify $\beta\eta$-equal terms.

Interpretation

Let S_1, S_2 two instances of STL.

An interpretation $\theta : S_1 \to S_2$ maps

- each base type b of S_1 to a type $\theta(b)$ of S_2, and
- each constant $c : \rho$ of S_1 to a closed term $\theta(c)$ in S_2 such that S_2 proves $\vdash \theta(c) : \theta(\rho)$.

For every S_1-type ρ, $\theta(\rho)$ is the S_2-type obtained by replacing every base type b by $\theta(b)$.

For every S_1-term M, $\theta(M)$ is the S_2-term obtained by replacing every base type b by $\theta(b)$ and every constant c by $\theta(c)$.

Interpretation

Let S_1, S_2 two instances of STL.

An interpretation $\theta : S_1 \to S_2$ maps

- each base type b of S_1 to a type $\theta(b)$ of S_2, and
- each constant $c : \rho$ of S_1 to a closed term $\theta(c)$ in S_2 such that S_2 proves $\vdash \theta(c) : \theta(\rho)$.

For every S_1-type ρ, $\theta(\rho)$ is the S_2-type obtained by replacing every base type b by $\theta(b)$.

For every S_1-term M, $\theta(M)$ is the S_2-term obtained by replacing every base type b by $\theta(b)$ and every constant c by $\theta(c)$.

Lemma (Fundamental lemma of interpretation)

Interpretation commutes with substitution, respects $\beta\eta$-equality, and preserves typing:

$$\theta(M[N/x]) = \theta(M)[\theta(N)/x].$$

If $M =_{\beta\eta} N$, *then* $\theta(M) =_{\beta\eta} \theta(N)$.

If S_1 *proves* $\Gamma \vdash M : \rho$, *then* S_2 *proves* $\theta(\Gamma) \vdash \theta(M) : \theta(\rho)$.

Church's Simple Theory of Types (CST, Church 1940)

CST is defined as the instance of STL given by

- a set $\mathcal{I}$ of base types for sets of individuals;
- The base type o (type of propositions, or truth values);
- the constants

$$\begin{array}{ccll} \supset & : & o \to o \to o & \quad (\to \text{ associates to the right}) \\ \forall_\rho & : & (\rho \to o) \to o & \text{ for every type } \rho \end{array}$$

Church's Simple Theory of Types (CST, Church 1940)

CST is defined as the instance of STL given by

- a set $\mathcal{I}$ of base types for sets of individuals;
- The base type o (type of propositions, or truth values);
- the constants

$$\begin{array}{lcll} \supset & : & o \to o \to o & \quad (\to \text{ associates to the right}) \\ \forall_\rho & : & (\rho \to o) \to o & \text{for every type } \rho \end{array}$$

Remark. Church used the constants

$$\begin{array}{lcll} \vee & : & o \to o \to o & \\ \exists_\rho & : & (\rho \to o) \to o & \\ \iota_\rho & : & (\rho \to o) \to \rho & \text{(choice)} \end{array}$$

with a *classical* Hilbert calculus to derive formulas (terms of type o)

Church's Simple Theory of Types (CST, Church 1940)

CST is defined as the instance of STL given by

- a set $\mathcal{I}$ of base types for sets of individuals;
- The base type o (type of propositions, or truth values);
- the constants

$$\begin{array}{lll} \supset & : & o \to o \to o \qquad (\to \text{ associates to the right}) \\ \forall_\rho & : & (\rho \to o) \to o \quad \text{for every type } \rho \end{array}$$

Remark. Church used the constants

$$\begin{array}{lll} \vee & : & o \to o \to o \\ \exists_\rho & : & (\rho \to o) \to o \\ \iota_\rho & : & (\rho \to o) \to \rho \quad \text{(choice)} \end{array}$$

with a *classical* Hilbert calculus to derive formulas (terms of type o)

We will work with *intuitionistic* natural deduction instead.

Intuitionistic natural deduction for CST

$$\begin{array}{rcl} A \supset B & := & \supset A\, B \\ \forall x : \rho\,.\, A(x) & := & \forall_\rho(\lambda x : \rho\,.\, A(x)) \end{array}$$

$$\frac{\Gamma \vdash \Delta, A : o}{\Delta, A \vdash_\Gamma A}\ \mathrm{Assm}$$

$$\frac{\Delta, A \vdash_\Gamma B}{\Delta \vdash_\Gamma A \supset B}\ \supset^+ \qquad \frac{\Delta \vdash_\Gamma A \supset B \qquad \Delta \vdash_\Gamma A}{\Delta \vdash_\Gamma B}\ \supset^-$$

$$\frac{\Delta \vdash_{\Gamma, x:\rho} A(x)}{\Delta \vdash_\Gamma \forall x : \rho\,.\, A(x)}\ \forall_\rho^+\ (*)$$

$$\frac{\Delta \vdash_\Gamma \forall x : \rho\,.\, A(x) \qquad \Gamma \vdash M : \rho}{\Delta \vdash_\Gamma A(M)}\ \forall_\rho^-$$

$(*)\ x \notin \mathrm{FV}(\Delta)$

RCST: CST with realizers

We extend CST to a system RCST by adding:

a new base type δ,

the constants

$$\begin{aligned} &\mathrm{Fun} : (\delta \to \delta) \to \delta \\ &\mathrm{app} : \delta \to \delta \to \delta \end{aligned}$$

RCST: CST with realizers

We extend CST to a system RCST by adding:

- a new base type δ,
- the constants

$$\mathrm{Fun} : (\delta \to \delta) \to \delta$$
$$\mathrm{app} : \delta \to \delta \to \delta$$

Terms of type δ are called *programs*.

RCST: CST with realizers

We extend CST to a system RCST by adding:

a new base type δ,

the constants

$$\mathrm{Fun} : (\delta \to \delta) \to \delta$$
$$\mathrm{app} : \delta \to \delta \to \delta$$

Terms of type δ are called *programs*.

We add the proof rule rule

$$\frac{\Gamma \vdash M : \delta \to \delta \qquad \Gamma \vdash N : \delta \qquad \Delta \vdash_\Gamma A(\mathrm{app}\,(\mathrm{Fun}\, M)\, N)}{\Delta \vdash_\Gamma A(M\, N)}\ \delta$$

RCST: CST with realizers

We extend CST to a system RCST by adding:

a new base type δ,

the constants

$$\begin{array}{l} \mathrm{Fun} : (\delta \to \delta) \to \delta \\ \mathrm{app} : \delta \to \delta \to \delta \end{array}$$

Terms of type δ are called *programs*.

We add the proof rule rule

$$\frac{\Gamma \vdash M : \delta \to \delta \qquad \Gamma \vdash N : \delta \qquad \Delta \vdash_\Gamma A(\mathrm{app}\,(\mathrm{Fun}\,M)\,N)}{\Delta \vdash_\Gamma A(M\,N)}\ \delta$$

Remark: Using the notation

$$\begin{array}{rcl} \lambda_\delta a\,.\,M(a) & := & \mathrm{Fun}(\lambda a : \delta\,.\,M(a)) \\ M{\cdot}N & := & \mathrm{app}\,M\,N \end{array}$$

the rule δ essentially says that $(\lambda_\delta a\,.\,M(a)){\cdot}N$ equals $M[N/a]$.

Interpretation of CST in RCST (realizability)

$\mathbf{R} : \mathrm{CST} \to \mathrm{RCST}$

$$
\begin{array}{rcl}
\mathbf{R}(\iota) & := & \iota \\
\mathbf{R}(o) & := & \delta \to o \\[1ex]
\mathbf{R}(\supset) & := & \lambda x, y : \delta \to o \,.\, \lambda d : \delta \,.\, \forall a : \delta \,.\, x\, a \supset y\,(d{\cdot}a) \\
\mathbf{R}(\forall_\rho) & := & \lambda p : \mathbf{R}(\rho) \to \delta \to o \,.\, \lambda a : \delta \,.\, \forall x : \mathbf{R}(\rho) \,.\, p\, x\, a
\end{array}
$$

Interpretation of CST in RCST (realizability)

$\mathbf{R} : \mathrm{CST} \to \mathrm{RCST}$

$$
\begin{aligned}
\mathbf{R}(\iota) &:= \iota \\
\mathbf{R}(o) &:= \delta \to o \\
\mathbf{R}(\supset) &:= \lambda x, y : \delta \to o \,.\, \lambda d : \delta \,.\, \forall a : \delta \,.\, x\, a \supset y\,(d \cdot a) \\
\mathbf{R}(\forall_\rho) &:= \lambda p : \mathbf{R}(\rho) \to \delta \to o \,.\, \lambda a : \delta \,.\, \forall x : \mathbf{R}(\rho) \,.\, p\, x\, a
\end{aligned}
$$

Remark: Using the notation $a\,\mathbf{r}\,A := \mathbf{R}(A)\,a$, RCST derives

$$
\begin{aligned}
\mathrm{Fun}(f)\,\mathbf{r}\,(A \supset B) &\supset\subset \forall a : \delta \,.\, (a\,\mathbf{r}\,A) \supset ((f\ a)\,\mathbf{r}\,B) \\
a\,\mathbf{r}\,\forall x : \rho \,.\, A(x) &\supset\subset \forall x : \mathbf{R}(\rho) \,.\, a\,\mathbf{r}\,A(x)
\end{aligned}
$$

Soundness

Theorem (B, Hou 2017)

From a CST *proof of* $\Delta \vdash_{\Gamma} A$ *one can extract a program* P *such that:*

$$\vec{b}\,\mathbf{r}\,\Delta \vdash_{\mathbf{R}(\Gamma),\vec{b}:\vec{\delta}} P\,\mathbf{r}\,A \text{ in } \mathrm{RCST}$$

Proof.

Induction on derivations.

The extracted program is obtained form the derivation, essentially, by interpreting the rules for implication by the constructors Fun and app and ignoring the rules for universal quantification (i.e. interpreting them by the identity).

$\square$

Proof terms

For the implementation of program extraction it is necessary to represent proofs by terms. These can be modelled as an instance CSTPR of STL that extends CST. One adds:

a new base type **pr**,

the constants

$$\begin{aligned}
\supset^+ &: o \to (\mathbf{pr} \to \mathbf{pr}) \to \mathbf{pr} \\
\supset^- &: \mathbf{pr} \to \mathbf{pr} \to \mathbf{pr} \\
\forall_\rho^+ &: (\rho \to \mathbf{pr}) \to \mathbf{pr} \\
\forall_\rho^- &: \mathbf{pr} \to \rho \to \mathbf{pr}
\end{aligned}$$

Proof terms

For the implementation of program extraction it is necessary to represent proofs by terms. These can be modelled as an instance CSTPR of STL that extends CST. One adds:

a new base type **pr**,

the constants

$$\begin{aligned}
&\supset^{+} : o \to (\mathbf{pr} \to \mathbf{pr}) \to \mathbf{pr}\\
&\supset^{-} : \mathbf{pr} \to \mathbf{pr} \to \mathbf{pr}\\
&\forall_{\rho}^{+} : (\rho \to \mathbf{pr}) \to \mathbf{pr}\\
&\forall_{\rho}^{-} : \mathbf{pr} \to \rho \to \mathbf{pr}
\end{aligned}$$

The natural deduction calculus now derives judgments of the form

$$\Delta \vdash_{\Gamma} d : A$$

where Δ is a finite set of assumptions $u_1 : A_1, \dots, u_n : A_n$, labelled by different assumption variables u_i that may occur in the proof d.

Proof rules with proof terms

$$\frac{\Gamma \vdash \vec{A}, A : o}{\vec{u} : A, u : A \vdash_\Gamma u : A}$$

$$\frac{\Delta, u : A \vdash_\Gamma d : B}{\Delta \vdash_\Gamma \supset^+ A\,(\lambda u : \mathbf{pr}\,.\,d) : A \supset B} \qquad \frac{\Delta \vdash_\Gamma d : A \supset B \qquad \Delta \vdash_\Gamma e : A}{\Delta \vdash_\Gamma \supset^- d\, e : B}$$

$$\frac{\Delta \vdash_{\Gamma, x:\rho} d : A(x)}{\Delta \vdash_\Gamma \forall^+_\rho(\lambda x : \rho\,.\,d) : \forall x : \rho\, A(x)} \qquad \frac{\Delta \vdash_\Gamma d : \forall x : \rho\, A(x) \qquad \Gamma \vdash M : \rho}{\Delta \vdash_\Gamma \forall^-_\rho\, d\, M : A(M)}$$

Program extraction

Program extraction is a *partial* interpretation
$\mathbf{ep} : \mathrm{CSTPR} \to \mathrm{RCST} \cup \{\bot\}$ defined by

$$\begin{array}{rcl}
\mathbf{ep}(\mathbf{pr}) & = & \delta \\
\mathbf{ep}(\supset^+) & = & \mathrm{Fun} \\
\mathbf{ep}(\supset^-) & = & \mathrm{app} \\
\mathbf{ep}(\forall^+_\rho) & = & \mathrm{id}_\delta \\
\mathbf{ep}(\forall^-_\rho) & = & \mathrm{id}_\delta
\end{array}$$

where $\mathrm{id}_\delta = \lambda a : \delta \,.\, a$ ($\mathbf{ep}(c) = \bot$ for other constants). Hence

$$\begin{array}{rcl}
\mathbf{ep}(u) & = & u \\
\mathbf{ep}(\supset^+ A\,(\lambda u : \mathbf{pr} \,.\, d)) & = & \lambda_\delta u \,.\, \mathbf{ep}(d) \\
\mathbf{ep}(\supset^- d\ e) & = & \mathbf{ep}(d) \cdot \mathbf{ep}(e) \\
\mathbf{ep}(\forall^+_\rho (\lambda x : \rho \,.\, d)) & = & \mathbf{ep}(d) \\
\mathbf{ep}(\forall^-_\rho\, d\ M) & = & \mathbf{ep}(d)
\end{array}$$

Soundness with proof terms

Theorem

If $\Delta \vdash_\Gamma d : A$ *in* CSTPR, *then,*

$$\mathbf{R}(\Delta) \vdash_{\mathbf{R}(\Gamma),\Delta^\delta} \mathbf{ep}(d)\,\mathbf{r}\,A \text{ in } \mathrm{RCST}$$

where

$$\mathbf{R}(\Delta) = \{u\,\mathbf{r}\,B \mid u : B \in \Delta\}$$
$$\Delta^\delta = \{u : \delta \mid u : B \in \Delta\}$$

Proof.

Induction on d.

If one extends RCST with proof terms, one can define the soundness proof by structural recursion on d:

$$\begin{array}{rcl} \mathbf{R}(u) & = & u' \quad (u' \text{ a fresh assumption variable}) \\ \mathbf{R}(\supset^+ A(\lambda u : \mathbf{pr}\,.\,d)) & = & \delta\,(\lambda u : \delta\,.\,\mathbf{ep}(d))\,u \\ & & \quad (\forall^+_\delta(\lambda a : \delta\,.\,(\supset^+(\mathbf{R}(A)\,a)\lambda u' : \mathbf{pr}\,.\,\mathbf{R}(d))) \\ \mathbf{R}(\supset^-\,d\,e) & = & \supset^-\,(\forall^-_\delta\,\mathbf{R}(d)\,\mathbf{ep}(d))\,\mathbf{R}(e) \\ & \ldots & \end{array}$$

Polymorphic types

We assign polymorphic types to (certain) programs of RCST.

The system is an instance of STL, known as **F**$\underline{\omega}$ (Barendregt 1992), whose types we call *kinds*.

Roughly speaking, **F**$\underline{\omega}$ is obtained from CST by erasing all individual types.

F$\underline{\omega}$ has

- one base kind $*$ (kind of types)
- the constants

$$\begin{array}{lcll} \Rightarrow & : & * \to * \to * & \\ \forall_\kappa & : & (\kappa \to *) \to * & \text{for every kind } \kappa \end{array}$$

Polymorphic types

We assign polymorphic types to (certain) programs of RCST.

The system is an instance of STL, known as **F**$\underline{\omega}$ (Barendregt 1992), whose types we call *kinds*.

Roughly speaking, **F**$\underline{\omega}$ is obtained from CST by erasing all individual types.

F$\underline{\omega}$ has

- one base kind $*$ (kind of types)
- the constants

$$\begin{array}{lcll} \Rightarrow & : & * \to * \to * & \\ \forall_\kappa & : & (\kappa \to *) \to * & \text{for every kind } \kappa \end{array}$$

Terms of kind $*$ are called *polymorphic types*. Terms of other kind are called *operators*.

Polymorphic type assignment

Let $\Delta' := \vec{b} : \Delta$

$$\frac{\Gamma \vdash \Delta, A : *}{\Delta', a : A \vdash_\Gamma a : A}$$

$$\frac{\Delta', a : A \vdash_\Gamma M : B}{\Delta' \vdash_\Gamma \lambda_\delta a \,.\, M : A \Rightarrow B} \qquad \frac{\Delta' \vdash_\Gamma M : A \Rightarrow B \qquad \Delta' \vdash_\Gamma N : A}{\Delta' \vdash_\Gamma M \cdot N : B}$$

$$\frac{\Delta' \vdash_{\Gamma, \alpha:\kappa} M : A\,\alpha}{\Delta' \vdash_\Gamma M : \forall_\kappa A} \; \alpha \text{ fresh} \qquad \frac{\Delta' \vdash_\Gamma M : \forall_\kappa A \qquad \Gamma \vdash B : \kappa}{\Delta' \vdash_\Gamma M : A\,B}$$

Polymorphic type assignment

Let $\Delta' := \vec{b} : \Delta$

$$\frac{\Gamma \vdash \Delta, A : *}{\Delta', a : A \vdash_\Gamma a : A}$$

$$\frac{\Delta', a : A \vdash_\Gamma M : B}{\Delta' \vdash_\Gamma \lambda_\delta a \,.\, M : A \Rightarrow B} \qquad \frac{\Delta' \vdash_\Gamma M : A \Rightarrow B \qquad \Delta' \vdash_\Gamma N : A}{\Delta' \vdash_\Gamma M{\cdot}N : B}$$

$$\frac{\Delta' \vdash_{\Gamma,\alpha:\kappa} M : A\,\alpha}{\Delta' \vdash_\Gamma M : \forall_\kappa A} \quad \alpha \text{ fresh} \qquad \frac{\Delta' \vdash_\Gamma M : \forall_\kappa A \qquad \Gamma \vdash B : \kappa}{\Delta' \vdash_\Gamma M : A\,B}$$

By a straightforward induction on $\mathbf{F}\underline{\omega}$ derivations one shows:

Lemma

If $\Delta' \vdash_\Gamma M : A$ in $\mathbf{F}\underline{\omega}$, then $\Gamma \vdash \Delta, A : *$ in $\mathbf{F}\underline{\omega}$, and $\vec{b} : \delta \vdash M : \delta$ in RCST.

The type of a formula

We define a partial interpretation $\mathcal{T} : \mathrm{CST} \to \mathbf{F}\underline{\omega} \cup \{\bot\}$:

$$\begin{array}{rcl} \mathcal{T}(o) & := & * \\ \mathcal{T}(\supset) & := & \Rightarrow \\ \mathcal{T}(\forall_\rho) & := & \begin{cases} \forall_{\mathcal{T}(\rho)} & \text{if } \rho = \vec{\sigma} \to o \\ \lambda A : * \,.\, A & \text{otherwise} \end{cases} \end{array}$$

The type of a formula

We define a partial interpretation $\mathcal{T} : \mathrm{CST} \to \mathbf{F}\underline{\omega} \cup \{\perp\}$:

$$
\begin{aligned}
\mathcal{T}(o) &:= * \\
\mathcal{T}(\supset) &:= \Rightarrow \\
\mathcal{T}(\forall_\rho) &:= \begin{cases} \forall_{\mathcal{T}(\rho)} & \text{if } \rho = \vec{\sigma} \to o \\ \lambda A : * \,.\, A & \text{otherwise} \end{cases}
\end{aligned}
$$

By the fundamental lemma of partial interpretation one has:

If $\Gamma \vdash M : \rho$ in CST and $\mathcal{T}(\rho) \neq \perp$, then $\mathcal{T}(\Gamma) \vdash \mathcal{T}(M) : \mathcal{T}(\rho)$.

I.p., for every CST formula $\Gamma \vdash A : o$ one has $\mathcal{T}(\Gamma) \vdash \mathcal{T}(A) : *$, i.e., $\mathcal{T}(A)$ is a polymorphic type in the context $\mathcal{T}(\Gamma)$.

The type of the extracted program

Theorem

If $\Delta \vdash_\Gamma d : A$ *in* CSTPR*, then,* $\mathcal{T}(\Gamma) \vdash \mathbf{ep}(d) : \mathcal{T}(A)$.

Proof.

Induction on d. $\square$

The missing logical operators

$$
\begin{aligned}
\wedge &:= \lambda x, y : o \,.\, \forall z : o \,.\, (x \supset y \supset z) \supset z \\
\vee &:= \lambda x, y : o \,.\, \forall z : o \,.\, (x \supset z) \supset (y \supset z) \supset z \\
\bot &:= \forall z : o \,.\, z \\
\exists_\rho &:= \lambda p : \rho \to o \,.\, \forall z : o \,.\, (\forall x : \rho \,.\, p\, x \supset z) \supset z
\end{aligned}
$$

The missing logical operators

$$
\begin{array}{rcl}
\wedge & := & \lambda x, y : o \,.\, \forall z : o \,.\, (x \supset y \supset z) \supset z \\
\vee & := & \lambda x, y : o \,.\, \forall z : o \,.\, (x \supset z) \supset (y \supset z) \supset z \\
\bot & := & \forall z : o \,.\, z \\
\exists_\rho & := & \lambda p : \rho \to o \,.\, \forall z : o \,.\, (\forall x : \rho \,.\, p\, x \supset z) \supset z
\end{array}
$$

Remark: These definitions can be easily found by observing that

$$A \quad \supset\subset \quad \forall z : o \,.\, (A \supset z) \supset z$$

Now, replace A by $x \wedge y$ etc., and apply well-known equivalences.

For example, $((x \wedge y) \supset z) \supset z \quad \supset\subset \quad (x \supset y \supset z) \supset z$.

Derived rules for $\wedge, \vee, \bot, \exists_\rho$

$$\frac{\Delta \vdash_\Gamma A \qquad \Delta \vdash_\Gamma B}{\Delta \vdash_\Gamma A \wedge B}\ \wedge^+ \qquad \frac{\Delta \vdash_\Gamma A \wedge B}{\Delta \vdash_\Gamma A}\ \wedge^-_{\mathrm{L}} \qquad \frac{\Delta \vdash_\Gamma A \wedge B}{\Delta \vdash_\Gamma B}\ \wedge^-_{\mathrm{R}}$$

$$\frac{\Delta \vdash_\Gamma A \qquad \Gamma \vdash B : o}{\Delta \vdash_\Gamma A \vee B}\ \vee^+_{\mathrm{L}} \qquad \frac{\Delta \vdash_\Gamma B \qquad \Gamma \vdash A : o}{\Delta \vdash_\Gamma A \vee B}\ \vee^+_{\mathrm{R}}$$

$$\frac{\Delta \vdash_\Gamma A \vee B \qquad \Delta, A \vdash_\Gamma C \qquad \Delta, B \vdash_\Gamma C}{\Delta \vdash_\Gamma C}\ \vee^-$$

$$\frac{\Delta \vdash_\Gamma \bot \qquad \Gamma \vdash A : o}{\Delta \vdash_\Gamma A}\ \bot^-$$

$$\frac{\Delta \vdash_\Gamma A(M) \qquad \Gamma \vdash M : \rho}{\Delta \vdash_\Gamma \exists x : \rho \,.\, A(x)}\ \exists^+_\rho$$

$$\frac{\Delta \vdash_\Gamma \exists x : \rho \,.\, A(x) \qquad \Delta, A(x) \vdash_{\Gamma, x:\rho} B}{\Delta \vdash_\Gamma B}\ \exists^-_\rho \qquad x \notin \mathrm{FV}(\Delta, B)$$

Equality

$$=_\rho \quad := \quad \lambda x, y : \rho \,.\, \forall p : \rho \to o \,.\, p\,x \supset p\,y \qquad \text{(Leibniz equality)}$$

Equality

$$=_\rho \quad := \quad \lambda x, y : \rho \,.\, \forall p : \rho \to o \,.\, p\, x \supset p\, y \qquad \text{(Leibniz equality)}$$

Alternatively, one can define equality as the intersection of all reflexive relations (similarly to equality in Martin-Löf type theory):

$$=_\rho' \quad := \quad \lambda x, y : \rho \,.\, \forall e : \rho \to \rho \to o \,.\, (\forall z : \rho \,.\, e\, z\, z) \supset e\, x\, y$$

Equality

$$=_\rho \quad := \quad \lambda x, y : \rho \,.\, \forall p : \rho \to o \,.\, p\, x \supset p\, y \qquad \text{(Leibniz equality)}$$

Alternatively, one can define equality as the intersection of all reflexive relations (similarly to equality in Martin-Löf type theory):

$$='_\rho \quad := \quad \lambda x, y : \rho \,.\, \forall e : \rho \to \rho \to o \,.\, (\forall z : \rho \,.\, e\, z\, z) \supset e\, x\, y$$

Exercise. Prove in CST:

(1) $=_\rho$ is an equivalence relation,

(2) $=_\rho$ and $='_\rho$ are equivalent.

Derived rules for equality

$$\frac{\Delta, p\,A \vdash_{\Gamma, p:\rho\to o} p\,B}{\Delta \vdash_{\Gamma} A =_{\rho} B} =^{+} \qquad p \notin \mathrm{FV}(\Delta, A, B)$$

$$\frac{\Delta \vdash_{\Gamma} A =_{\rho} B \qquad \Delta \vdash_{\Gamma} P\,A}{\Delta \vdash_{\Gamma} P\,B} =^{-}$$

Least and greatest fixed points

For every type $\rho = \vec{\sigma} \to o$ $(= \sigma_1 \to \ldots \to \sigma_n \to o)$ we set

$$
\begin{aligned}
\subseteq_\rho \quad &:= \quad \lambda P, Q : \rho \,.\, \forall \vec{x} : \vec{\sigma} \,.\, P\,\vec{x} \supset Q\,\vec{x} \\
\overset{\vee}{\Phi} \quad &:= \quad \lambda P : \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \exists Y : \rho \,.\, Y \subseteq_\rho P \wedge \Phi\, Y\, \vec{x} \\
\overset{\wedge}{\Phi} \quad &:= \quad \lambda P : \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \forall Y : \rho \,.\, P \subseteq_\rho Y \supset \Phi\, Y\, \vec{x}
\end{aligned}
$$

and define

$$
\begin{aligned}
\mu_\rho \quad &:= \quad \lambda \Phi : \rho \to \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \forall P : \rho \,.\, (\overset{\vee}{\Phi}\, P \subseteq_\rho P) \supset P\vec{x} \\
\nu_\rho \quad &:= \quad \lambda \Phi : \rho \to \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \exists P : \rho \,.\, (P \subseteq_\rho \overset{\wedge}{\Phi}\, P) \wedge P\vec{x}
\end{aligned}
$$

Least and greatest fixed points

For every type $\rho = \vec{\sigma} \to o$ $(= \sigma_1 \to \ldots \to \sigma_n \to o)$ we set

$$\begin{aligned}
\subseteq_\rho &:= \lambda P, Q : \rho \,.\, \forall \vec{x} : \vec{\sigma} \,.\, P\,\vec{x} \supset Q\,\vec{x} \\
\overset{\vee}{\Phi} &:= \lambda P : \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \exists Y : \rho \,.\, Y \subseteq_\rho P \wedge \Phi\, Y\, \vec{x} \\
\overset{\wedge}{\Phi} &:= \lambda P : \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \forall Y : \rho \,.\, P \subseteq_\rho Y \supset \Phi\, Y\, \vec{x}
\end{aligned}$$

and define

$$\begin{aligned}
\mu_\rho &:= \lambda \Phi : \rho \to \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \forall P : \rho \,.\, (\overset{\vee}{\Phi}\, P \subseteq_\rho P) \supset P\vec{x} \\
\nu_\rho &:= \lambda \Phi : \rho \to \rho \,.\, \lambda \vec{x} : \vec{\sigma} \,.\, \exists P : \rho \,.\, (P \subseteq_\rho \overset{\wedge}{\Phi}\, P) \wedge P\vec{x}
\end{aligned}$$

Informally:

$$\overset{\vee}{\Phi}\, P = \bigcup_{Y \subseteq_\rho P} \Phi\, Y \qquad \mu_\rho\, \Phi = \bigcap_{\overset{\vee}{\Phi}\, P \subseteq_\rho P} P$$

$$\overset{\wedge}{\Phi}\, P = \bigcap_{P \subseteq_\rho Y} \Phi\, Y \qquad \nu_\rho\, \Phi = \bigcup_{P \subseteq_\rho \overset{\wedge}{\Phi}\, P} P$$

Aczel's rule sets (Aczel 1977)

An operator $\Phi : (\sigma \to o) \to (\sigma \to o)$ can also be viewed (via "uncurrying") as

$$\Phi : (\sigma \to o) \times \sigma \to o$$

Thus, Φ is the set of pairs (X, x) where $\Phi\ X\ x$, i.e. $x \in \Phi\, X$. Aczel calls the pairs (X, x) *rules*.

He calls a set P *closed* if whenever $(X, x) \in \Phi$ and $X \subseteq P$, then $x \in P$.

This is the same as saying $\overset{\vee}{\Phi}\, P \subseteq P$.

Aczel defines a set $I(\Phi)$ inductively from Φ as the least closed set.

Hence $I(\Phi) = \mu_\rho\, \Phi$.

Derived general fixed point rules

$$\frac{\Delta \vdash_\Gamma P \subseteq_\rho \mu_\rho \Phi}{\Delta \vdash_\Gamma \Phi\, P \subseteq_\rho \mu_\rho \Phi}\ \mu_\rho^+$$

$$\frac{\Delta, x \subseteq_\rho P \vdash_\Gamma \Phi\, x \subseteq_\rho P}{\Delta \vdash_\Gamma \mu_\rho \Phi \subseteq_\rho P}\ \mu_\rho^- \qquad x \notin \mathrm{FV}(\Delta, P, \Phi)$$

$$\frac{\Delta \vdash_\Gamma \nu_\rho \Phi \subseteq_\rho P}{\Delta \vdash_\Gamma \nu_\rho \Phi \subseteq_\rho \Phi\, P}\ \nu_\rho^+$$

$$\frac{\Delta, P \subseteq_\rho x \vdash_\Gamma P \subseteq_\rho \Phi\, x}{\Delta \vdash_\Gamma P \subseteq_\rho \nu_\rho \Phi}\ \nu_\rho^- \qquad x \notin \mathrm{FV}(\Delta, P, \Phi)$$

Monotone operators

An operator $\Phi : \rho \to \rho$ is *monotone* if it preserves inclusion:

$$\mathrm{Mon}_\rho\, \Phi := \forall x, y : \rho \,.\, x \subseteq_\rho y \supset \Phi\, x \subseteq_\rho \Phi\, y$$

Clearly, if Φ is monotone, then

$$\Phi\, x \approx_\rho \overset{\vee}{\Phi}\, x \approx_\rho \overset{\wedge}{\Phi}\, x$$

where $x \approx_\rho y := x \subseteq_\rho y \wedge y \subseteq_\rho x$, and therefore

$$\mu_\rho \Phi \approx_\rho \bigcap_{\Phi\, x \subseteq_\rho x} x$$

$$\nu_\rho \Phi \approx_\rho \bigcup_{x \subseteq_\rho \Phi\, x} x$$

Derived fixed point rules for monotone operators

$$\frac{\Gamma \vdash \Phi : \rho \to \rho}{\Delta \vdash_\Gamma \Phi(\mu_\rho \Phi) \subseteq_\rho \mu_\rho \Phi}\,\mathrm{Cl}_\rho$$

$$\frac{\Delta \vdash_\Gamma \mathrm{Mon}_\rho\,\Phi \qquad \Delta \vdash_\Gamma \Phi\, P \subseteq_\rho P}{\Delta \vdash_\Gamma \mu_\rho \Phi \subseteq_\rho P}\,\mathrm{MInd}_\rho$$

$$\frac{\Gamma \vdash \Phi : \rho \to \rho}{\Delta \vdash_\Gamma \nu_\rho \Phi \subseteq_\rho \Phi(\nu_\rho \Phi)}\,\mathrm{CoCl}_\rho$$

$$\frac{\Delta \vdash_\Gamma \mathrm{Mon}_\rho\,\Phi \qquad \Delta \vdash_\Gamma P \subseteq_\rho \Phi\, P}{\Delta \vdash_\Gamma P \subseteq_\rho \nu_\rho \Phi}\,\mathrm{MCoInd}_\rho$$

Definitional extensions of CST

To simplify extracted programs we add the defined logical operators discussed earlier as constants to CST together with proof constants for the derived logical rules.

Definitional extensions of CST

To simplify extracted programs we add the defined logical operators discussed earlier as constants to CST together with proof constants for the derived logical rules.

Similarly, we add to RCST definable constants

$$
\begin{array}{rcl}
\mathrm{Nil} & : & \delta \\
\mathrm{pr_L}, \mathrm{pr_R}, \mathrm{L}, \mathrm{R} & : & \delta \to \delta \\
(\mathrm{app}), \mathrm{Pair} & : & \delta \to \delta \to \delta \\
(\mathrm{Fun}), \mathrm{rec} & : & (\delta \to \delta) \to \delta \\
\mathrm{case} & : & \delta \to (\delta \to \delta) \to (\delta \to \delta) \to \delta
\end{array}
$$

Extended realizability

$$
\begin{aligned}
\mathbf{R}(\iota) &:= \iota \\
\mathbf{R}(o) &:= \delta \to \iota \\[1ex]
\mathbf{R}(\supset) &:= \lambda A, B : \delta \to o \,.\, \lambda d : \delta. \\
&\qquad \exists f : \delta \to \delta \,.\, d =_\delta \mathrm{Fun}\, f \wedge \forall a : \delta \,.\, A\, a \supset B(f\ a) \\
\mathbf{R}(\wedge) &:= \lambda A, B : \delta \to o \,.\, \lambda d : \delta \,. \\
&\qquad \exists a, b : \delta \,.\, d =_\delta \mathrm{Pair}\, a\, b \wedge A\, a \wedge B\, b \\
\mathbf{R}(\vee) &:= \lambda A, B : \delta \to o . \lambda d : \delta \,. \\
&\qquad \exists a : \delta \ .\, (d =_\delta \mathrm{L}\, a \wedge A\, a) \vee (d =_\delta \mathrm{R}\, a \wedge B\, a) \\
\mathbf{R}(\forall_\rho) &:= \lambda A : \mathbf{R}(\rho) \to \delta \to o \,.\, \lambda d : \delta \,.\, \forall x : \mathbf{R}(\rho)\ .\, A\, x\, d \\
\mathbf{R}(\exists_\rho) &:= \lambda A : \mathbf{R}(\rho) \to \delta \to o \,.\, \lambda d : \delta \,.\, \exists x : \mathbf{R}(\rho)\ .\, A\, x\, d \\
\mathbf{R}(=_\rho) &:= \lambda x, y : \mathbf{R}(\rho)\, \lambda d : \delta \,.\, x =_{\mathbf{R}(\rho)} y \\
\mathbf{R}(\mu_\rho) &:= \mu_{\mathbf{R}(\rho)} \\
\mathbf{R}(\nu_\rho) &:= \nu_{\mathbf{R}(\rho)}
\end{aligned}
$$

Extended program extraction: Propositional logic

$$
\begin{array}{rcl}
\supset^{+} & : & o \to (\mathbf{pr} \to \mathbf{pr}) \to \mathbf{pr} \\
\supset^{-} & : & \mathbf{pr} \to \mathbf{pr} \to \mathbf{pr} \\
\wedge^{+} & : & \mathbf{pr} \to \mathbf{pr} \to \mathbf{pr} \\
\wedge^{-}_{\mathrm{L}}, \wedge^{-}_{\mathrm{R}} & : & \mathbf{pr} \to \mathbf{pr} \\
\vee^{+}_{\mathrm{L}}, \vee^{+}_{\mathrm{R}} & : & \mathbf{pr} \to o \to \mathbf{pr} \\
\vee^{-} & : & \mathbf{pr} \to (\mathbf{pr} \to \mathbf{pr}) \to (\mathbf{pr} \to \mathbf{pr}) \to \mathbf{pr} \\
\mathbf{ep}(\mathbf{pr}) & := & \delta \\
\mathbf{ep}(\supset^{+}) & := & \mathrm{Fun} \\
\mathbf{ep}(\supset^{-}) & := & \mathrm{app} \\
\mathbf{ep}(\wedge^{+}) & := & \mathrm{Pair} \\
\mathbf{ep}(\wedge^{-}_{\mathrm{L}}) & := & \mathrm{pr}_{\mathrm{L}} \\
\mathbf{ep}(\wedge^{-}_{\mathrm{R}}) & := & \mathrm{pr}_{\mathrm{R}} \\
\mathbf{ep}(\vee^{+}_{\mathrm{L}}) & := & \mathrm{L} \\
\mathbf{ep}(\vee^{+}_{\mathrm{R}}) & := & \mathrm{R} \\
\mathbf{ep}(\vee^{-}) & := & \mathrm{case}
\end{array}
$$

Extended program extraction: Quantifiers and equality

$$
\begin{array}{rcl}
\forall^+_\rho & : & (\rho \to \mathbf{pr}) \to \mathbf{pr} \\
\forall^-_\rho & : & \mathbf{pr} \to \rho \to \mathbf{pr} \\
\exists^+_\rho & : & \mathbf{pr} \to \rho \to \mathbf{pr} \\
\exists^-_\rho & : & \mathbf{pr} \to (\rho \to \mathbf{pr}) \to \mathbf{pr} \\
=^+_\rho & : & (o \to \mathbf{pr}) \to \mathbf{pr} \\
=^-_\rho & : & \mathbf{pr} \to \mathbf{pr} \to \mathbf{pr}
\end{array}
$$

$$
\begin{array}{rcl}
\mathbf{ep}(\forall^+_\rho) = \mathbf{ep}(\forall^-_\rho) & := & \mathrm{id}_\delta \\
\mathbf{ep}(\exists^+_\rho) = \mathbf{ep}(\exists^-_\rho) & := & \mathrm{id}_\delta \\
\mathbf{ep}(=^+_\rho) & := & \lambda a : \delta \,.\, \mathrm{Nil} \\
\mathbf{ep}(=^-_\rho) & := & \lambda a, b : \delta \,.\, b
\end{array}
$$

Extended program extraction: Monotone (co)induction

$$
\begin{array}{rcl}
\mathrm{Cl}_\rho, \mathrm{CoCl}_\rho & : & (\rho \to \rho) \to \mathbf{pr} \\
\mathrm{MInd}_\rho, \mathrm{MCoInd}_\rho & : & \mathbf{pr} \to \mathbf{pr} \to \mathbf{pr} \\
\\
\mathbf{ep}(\mathrm{Cl}) = \mathbf{ep}(\mathrm{CoCl}) & := & \mathrm{Fun}\,\mathrm{id}_\delta \\
\mathbf{ep}(\mathrm{MInd}) & := & \lambda m, s : \delta \,.\, \mathrm{rec}(\lambda f : \delta \,.\, s \,\mathbf{o}\, (m \cdot f)) \\
\mathbf{ep}(\mathrm{MCoInd}) & := & \lambda m, s : \delta \,.\, \mathrm{rec}(\lambda f : \delta \,.\, (m \cdot f) \,\mathbf{o}\, s) \\
\text{where} & & \\
a \,\mathbf{o}\, b & := & \lambda_\delta c \,.\, a \cdot (b \cdot c)
\end{array}
$$

Extended soundness and typing

The Soundness Theorem holds for extended program extraction.

Similarly, the typing theorem holds for extended program extraction, w.r.t. a suitable extension of system $\mathbf{F}\underline{\omega}$ with the constructors

unit type	$\mathbf{1} : *$
functions, products and sums	$\Rightarrow, \otimes, \oplus : * \to * \to *$
universal quantifier	$\forall_\kappa : (\kappa \to *) \to *$
fixed points	$\mathbf{fix} : (* \to *) \to *$

Extended soundness and typing

The Soundness Theorem holds for extended program extraction.

Similarly, the typing theorem holds for extended program extraction, w.r.t. a suitable extension of system $\mathbf{F}\underline{\omega}$ with the constructors

unit type	$\mathbf{1} : *$
functions, products and sums	$\Rightarrow, \otimes, \oplus : * \to * \to *$
universal quantifier	$\forall_\kappa : (\kappa \to *) \to *$
fixed points	$\mathbf{fix} : (* \to *) \to *$

Remarks. To prove soundness for monotone induction and coinduction, the corresponding monotone principles in RCST are not sufficient since the realizability interpretation of a monotone operator need not be monotone. The general fixed point rules are needed.

Extended soundness and typing

The Soundness Theorem holds for extended program extraction.

Similarly, the typing theorem holds for extended program extraction, w.r.t. a suitable extension of system $\mathbf{F}\underline{\omega}$ with the constructors

unit type	$\mathbf{1} : *$
functions, products and sums	$\Rightarrow, \otimes, \oplus : * \to * \to *$
universal quantifier	$\forall_\kappa : (\kappa \to *) \to *$
fixed points	$\mathbf{fix} : (* \to *) \to *$

Remarks. To prove soundness for monotone induction and coinduction, the corresponding monotone principles in RCST are not sufficient since the realizability interpretation of a monotone operator need not be monotone. The general fixed point rules are needed.

In current systems, induction and coinduction require *strictly positive* operators.

Monotone fixed point in lambda calculus where studied in (Mendler 91) and (Matthes, Uustalu 2005).

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.
- Realizability and program extraction are defined as interpretations.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.
- Realizability and program extraction are defined as interpretations.
- Soundness guarantees correctness of extracted programs.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.
- Realizability and program extraction are defined as interpretations.
- Soundness guarantees correctness of extracted programs.
- Extracted programs are untyped, but are assigned types in $\mathbf{F}\underline{\omega}$.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.
- Realizability and program extraction are defined as interpretations.
- Soundness guarantees correctness of extracted programs.
- Extracted programs are untyped, but are assigned types in $\mathbf{F}\underline{\omega}$.
- Least and greatest fixed points of monotone operators can be defined, but are introduced as constants to simplify extracted programs.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.
- Realizability and program extraction are defined as interpretations.
- Soundness guarantees correctness of extracted programs.
- Extracted programs are untyped, but are assigned types in $\mathbf{F}\underline{\omega}$.
- Least and greatest fixed points of monotone operators can be defined, but are introduced as constants to simplify extracted programs.
- We don't compute with *proofs*. Computation with *programs* will be explained in Lecture 3 and in Hideki Tsuiki's lectures.

Summary of Lecture 1

- The simply typed lambda calculus (STL) and the notion of interpretation serve a a logical framework.
- Intuitionistic higher-order logic is an instance of STL.
- Realizability and program extraction are defined as interpretations.
- Soundness guarantees correctness of extracted programs.
- Extracted programs are untyped, but are assigned types in $\mathbf{F}\underline{\omega}$.
- Least and greatest fixed points of monotone operators can be defined, but are introduced as constants to simplify extracted programs.
- We don't compute with *proofs*. Computation with *programs* will be explained in Lecture 3 and in Hideki Tsuiki's lectures.
- Simplification of programs will be explained in Lecture 3.