Computational Logic

Interactive Theorem Proving

Week 5

Cezary Kaliszyk (VO)
Vincent van Oostrom (PS)

Nov 4, 2016

Summary

So far

Proof Assistants, HOL Light, $\lambda_{\rightarrow}$, Gentzen-style, Tactics

- Properties and limitations of $\lambda_{\rightarrow}$
- Induction and recursion in HOL

Today

- λ_P, its properties, Curry Howard
- HOL Light more advanced tactics

Dependent types vs Polymorphism vs CoC

Printf

What type does it have?

Bit-strings of length n

- Type of bit-strings: $bs : \mathbb{N} \to \star$
- Bit-string made of zeros: $0_{bs} : (\forall n : \mathbb{N}) bs(n)$
- $\mathbb{R}^{\mathbb{N}}$

Vectors (later polymorphic)

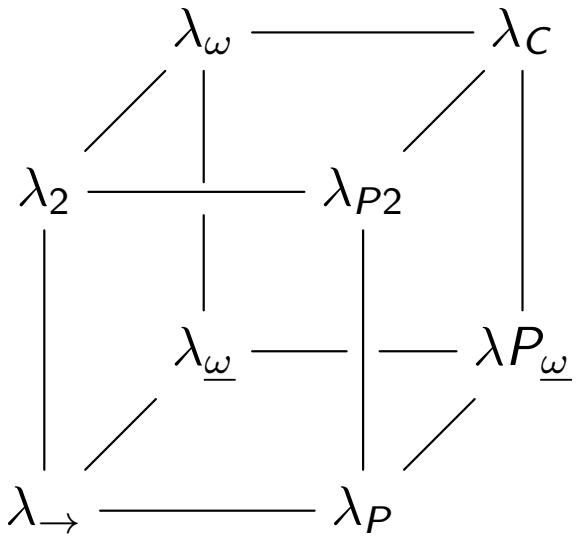
- Type of hd?

Constructive Division

$$a/b/\!\!/P \qquad \approx \qquad \frac{a}{b \neq 0} \quad .$$

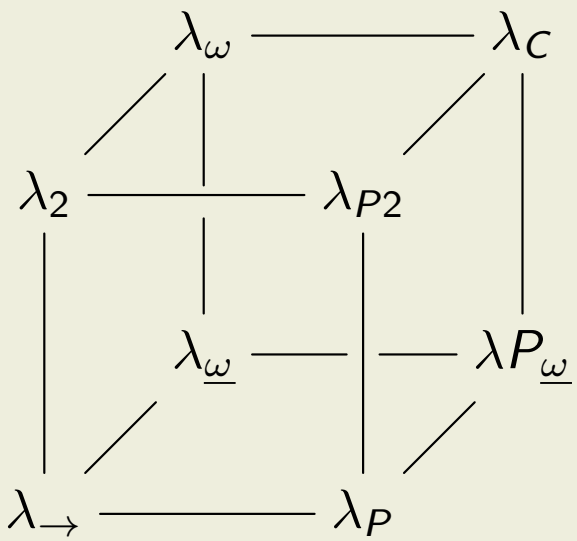
Lambda cube (Barendregt, 1991)

propositional logic	$\longleftrightarrow$	$\lambda_{\rightarrow}$
predicate logic	$\longleftrightarrow$	λ_P (dependent types)
2nd order propositional logic	$\longleftrightarrow$	λ_2, System F (2nd order typed λ-calc)
		$\lambda_{\underline{\omega}}$ (type operators)



Lambda cube (again)

Four kinds of dependencies



- terms on terms (already in $\lambda_\rightarrow$)
- dependent types (λ_P)
- polymorphism (λ_2)
- terms depend on types
- Combining the three is hard! (Girard's paradox)

Intuition behind λ_P

functions from A to B

$$A \to B$$

dependent functions from A to B

$$\Pi x : A.B$$

- Also called: dependent product
- Type of B can now depend on the argument x
- arrow type becomes a special case of dependent product

Three kinds of judgements

Kind formation judgements

$$\Gamma \vdash k : \Box$$

Kinding judgements

$$\Gamma \vdash \varphi : k$$

Typing judgements

$$\Gamma \vdash M : \tau$$

The meaning of $k : \Box$ is that k is a well-formed kind.

Syntax of λ_P

- variables

$$x, y, z, \ldots$$

- abstraction

$$\lambda x : M.N$$

- function application

$$MN$$

- dependent product

$$\Pi x : M.N \text{ (sometimes } \forall x : M.N)$$

- two sorts

$$\star, \Box$$

Abbreviations

If x is not free in k

We write $\tau \Rightarrow k$ instead of $(\Pi x : \tau)k$.

If x is not free is σ

We write $\tau \rightarrow \sigma$ instead of $(\forall x : \tau)\sigma$.

β-reduction in λ_P

- Like in $\lambda_\to$

$$(\lambda x : \tau . M) N \to_\beta M[x := N]$$

- Under lambda and application
- In the type of a λ-expression or Π-expression
- In a type application or under a Π

Rules of λ_P (1/3)

Axiom rule

$$\frac{}{\vdash \star : \Box}$$

Variable rule (condition: $x \notin \Gamma$)

$$\frac{\Gamma \vdash A : \{\star, \Box\}}{\Gamma, x : A \vdash x : A}$$

Weakening rule (condition: $x \notin \Gamma$)

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : \{\star, \Box\}}{\Gamma, x : C \vdash A : B}$$

Rules of λ_P (2/3)

Dependent product rule (same $\star$ or $\Box$)

$$\frac{\Gamma \vdash A : \star \qquad \Gamma, x : A \vdash B : \{\star, \Box\}}{\Gamma \vdash \Pi x : A.\, B : \{\star, \Box\}}$$

Abstraction rule

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : \{\star, \Box\}}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

Application Rule

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

Rules of λ_P (3/3)

Conversion Rule

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : \{\star, \Box\}}{\Gamma \vdash A : B'} \quad \text{where } B =_\beta B'$$

Examples

- $$X : *,\ x : X \vdash x : X$$
- $$X : * \vdash (X \to X) : *$$
- $$A : *,\ P : A \to *,\ a : A \vdash (P\,a) \to * : \Box$$
- $$\alpha : 0 \to \star, \beta : 0 \to \star \vdash \lambda y : (\forall x : 0.\alpha x \to \beta x).\lambda z : (\forall x : 0.\alpha x).\lambda x : 0.yx(zx) : (\forall x : 0.\alpha x \to \beta x) \to (\forall x : 0.\alpha x) \to \forall x : 0.\beta x$$
- $$\alpha : 0 \Rightarrow \star \vdash (\Pi y : 0)(\alpha y \Rightarrow \star) : \Box$$

Properties of λ_P

- Possible to extend by an existential quantifier
  - Disjoint union (coproduct) of types
- Strong Normalization (using forgetting map)
- Church-Rosser (corollary)
- Subject Reduction
- Type reconstruction is decidable in PTIME.
  - Type checking is undecidable!
- Type inhabitation in λ_P is undecidable
  - ?

Properties of λ_P

- Possible to extend by an existential quantifier
  - Disjoint union (coproduct) of types
- Strong Normalization (using forgetting map)
- Church-Rosser (corollary)
- Subject Reduction
- Type reconstruction is decidable in PTIME.
  - Type checking is undecidable!
- Type inhabitation in λ_P is undecidable

  First order intuitionistic logic is undecidable

Correspondence with first order logic

FOL corresponds to a fairly weak fragment of λ_P

- Only one type variable 0 (constant, type of individuals)
- All kinds are of the form $0 \Rightarrow \ldots \Rightarrow 0 \Rightarrow \star$
- Function symbols are distinguished object variables $0 \to \ldots \to 0$
- Constants are distinguished variables of type 0
- Other declarations may only be of the form $x : 0$

Proof correspondence

- Proof by generalization
  - abstraction $\lambda x : 0.M^{\varphi}$
- Proof by MP
  - application $M^{\forall x:0\varphi} N^0$

More: Proseminar.

Rules, Conversions, Tactics

- Rule: `thm -> thm`
  - Examples?
- Conversion: `term -> thm`
  - Examples?
- Tactic: goalstate refinement
  - Type?
- Thm_tactical: `thm_tactic -> thm_tactic`
- How to combine them?
  - THEN...
  - DEPTH_CONV
- How to transform one to other?
  - CONV_TAC, CONV_RULE

Rules, Conversions, Tactics

- Rule: `thm -> thm`
  - Examples?
- Conversion: `term -> thm`
  - Examples? `BETA_CONV`
- Tactic: goalstate refinement
  - Type?
- Thm_tactical: `thm_tactic -> thm_tactic`
- How to combine them?
  - THEN...
  - DEPTH_CONV
- How to transform one to other?
  - CONV_TAC, CONV_RULE

Rules, Conversions, Tactics

- Rule: `thm -> thm`
  - Examples?
- Conversion: `term -> thm`
  - Examples? `BETA_CONV`
- Tactic: goalstate refinement
  - Type? `goal -> goalstate`
- Thm_tactical: `thm_tactic -> thm_tactic`
- How to combine them?
  - THEN...
  - DEPTH_CONV
- How to transform one to other?
  - CONV_TAC, CONV_RULE

More advanced tactics

Tautologies

```
ITAUT `(A ==> B) ==> A ==> B`;;
TAUT `(p <=> (q <=> r)) <=> ((p <=> q) <=> r)`;;
```

Rewriting

- Matching
  - Slightly more general than first-order
- REWR_CONV does rewriting on top level

```
ADD_ASSOC;;
⊢ ∀mnp.m+n+p=(m+n)+p
REWR_CONV    ADD_ASSOC    `x+y+z`;;
⊢ x+y+z=(x+y)+z
```

- Conditional rewriting: REWRITE_RULE, REWRITE_CONV, REWRITE_TAC
- With basic rules: SIMP...

Summary

Today

- Properties of λ_P
- Curry Howard for λ_P
- More on HOL Light

Next time

- Polymorphism
- Set Theory Introduction