MATH230: Tutorial Twelve

Solutions

> Key ideas
> - Write recursive processes in λ-calculus,
> - Write programs in simple type theory,
> - Interpret programs as proofs of propositions,
> - Prove that certain types are uninhabited.
>
> Relevant lectures: Lambda Calculus and Typed Lambda Calculus Slides
> Relevant reading: Type Theory and Functional Programming, Simon Thompson
>
> Hand in exercises: 1b, 1d, 4a, 4b, 4c, 6
> **Due before the exam to the lecturer.**

Discussion Questions

- Write a program of the specified type in the given context:

$$p : A \times (B \times C) \;\vdash\; ? : (A \times B) \times C$$

$$\dfrac{\dfrac{\dfrac{p : A \times (B \times C)}{\mathsf{FST}\ p \ :\ A}\ \mathsf{FST} \quad \dfrac{\dfrac{p : A \times (B \times C)}{\mathsf{SND}\ p \ :\ B \times C}\ \mathsf{SND}}{\mathsf{FST}\ (\mathsf{SND}\ p)\ :\ B}\ \mathsf{FST}}{(\mathsf{FST}\ p,\ \mathsf{FST}\ (\mathsf{SND}\ p))\ :\ A \times B}\ \times \quad \dfrac{\dfrac{p\ :\ A \times (B \times C)}{\mathsf{SND}\ p\ :\ B \times C}\ \mathsf{SND}}{\mathsf{SND}\ (\mathsf{SND}\ p)\ :\ C}\ \mathsf{SND}}{((\mathsf{FST}\ p,\ \mathsf{FST}\ (\mathsf{SND}\ p)),\ \mathsf{SND}\ (\mathsf{SND}\ p))\ :\ (A \times B) \times C}\ \times$$

  This typing derivation shows the λ-term

$$((\mathsf{FST}\ p,\ \mathsf{FST}\ (\mathsf{SND}\ p)),\ \mathsf{SND}\ (\mathsf{SND}\ p))\ : (A \times B) \times C$$

  inhabits the stated type in the given context. Compare this to the natural deduction proof of the sequent

$$A \wedge (B \wedge C) \;\vdash\; (A \wedge B) \wedge C$$

- Determine some steps towards writing a program (λ-term) representing the unary function, INT-SQRT, that returns the greatest natural number whose square is less than or equal to the input.

  Solution:

  The integer square root of a natural number n is defined as the largest natural number x such that $x^2 \leq n$. We can search for this by starting at $t = 0$ and checking the condition $t^2 > n$, incrementing t by one until such a t is found. At which point the procedure should return $t - 1$.

  In order to code such a procedure in the λ-calculus we should use the Y combinator to recursively call a function. Following the process described in class we define a helper function which the Y combinator will recursively call.

  The first abstraction $\lambda s.$ is for the procedure to call itself.

  The second abstraction $\lambda t.$ is the abstraction we pass the test $t = 0$ to. Finally, the third abstraction is the number whose integer square root is to be computed.

  $$\begin{aligned} \mathsf{GO} :\equiv \ \lambda s.\lambda t.\lambda n.\ \mathsf{COND}\ & (>?\ (\mathsf{MULT}\ t\ t)\ n) \\ & (\mathsf{PRED}\ t) \\ & (s\ (\mathsf{SUCC}\ t)\ n) \end{aligned}$$

  Notice that third argument to COND calls the procedure with t incremented and n left unchanged. This is how the procedure moves on to test the next natural number.

  Together with the Y combinator and starting with $t = 0$ we define the procedure:

  $$\mathsf{INT\text{-}SQRT} :\equiv \mathsf{Y}\ \mathsf{GO}\ \mathsf{ZERO}$$

  Compute INT-SQRT FOUR to test this procedure.

Tutorial Exercises

1. Write recursive λ-expressions that represent the following functions of natural numbers. For each function determine an appropriate helper-function GO to put through the Y combinator.

   (a) SUM of two natural numbers

   Solution:

   (b) MULTiply two natural numbers

   Solution:

$$\text{GO} :\equiv \lambda s.\lambda m.\lambda n.\ \text{COND}\ (\text{ZERO?}\ b)\ \text{ZERO}\ (\text{SUM}\ a\ (s\ a\ (\text{PRED}\ b)))$$

$$\text{MULT} :\equiv \text{Y GO}$$

   (c) EXPONentiation of a base to an exponent

   Solution: Here we use the abstraction b for the base of the exponentiation and the abstraction over e for the exponent of the exponentiation.

$$\text{GO} :\equiv \lambda s.\lambda b.\lambda e.\ \text{COND}\ (\text{ZERO?}\ e)\ \text{ONE}\ (\text{MULT}\ b\ (s\ b\ (\text{PRED}\ e)))$$

$$\text{EXP} :\equiv \text{Y GO}$$

   (d) FACTorial of a natural number

   Solution:

   (e) INT-SQRT the smallest integer whose square is greater than input

   Solution: See discussion above for the derivation of this lambda encoding.

$$\begin{aligned} \text{GO} :\equiv\ & \lambda s.\lambda t.\lambda n.\ \text{COND}\ (>?\ (\text{MULT}\ t\ t)\ n) \\ & \qquad\qquad\qquad (\text{PRED}\ t) \\ & \qquad\qquad\qquad (s\ (\text{SUCC}\ t)\ n) \end{aligned}$$

$$\text{INT-SQRT} :\equiv \text{Y GO ZERO}$$

   (f) Calculate the nth FIBonacci number (Challenge!)

   Solution:

2. Write a λ-expression that can be used to compute the smallest natural number that satisfies a given unary-predicate $P?(x)$ that is represented by some λ-expression.

   Solution:

$$\text{GO} :\equiv \lambda s.\lambda n.\ \text{COND}\ (P?\ n)\ n\ (s\ (\text{SUCC}\ n))$$

   Combining this with the Y combinator yields a procedure μ that searches for the smallest natural number satifying the predicate P?

$$\mu :\equiv \text{Y GO ZERO}$$

3. (Challenge!) Represent the following processes in the λ-calculus to get an expression that can be used to test whether a natural number is prime. For simplicity, assume the input is greater than TWO.

   (a) REMAINDER calculate the remainder of a division.

   (b) DIVIDES? binary predicate does second divide first?

   (c) Implement bounded-search to satisfy a predicate.

   (d) PRIME? Unary-predicate to detect primality.

Curry-Howard Correspondence

4. Each problem below is of the form:

$$\Sigma \ \vdash ? : \mathsf{TYPE}$$

To answer the question you must provide a λ-term of the specified TYPE from the context Σ stated in the problem. To ensure the term is of the specified type you must use the typing rules for the construction and destruction of types.

Comment: In order to save space and help readability, the abstractions have been written without explicit typing. These can be inferred from the type of the entire term.

(a) $f : (A \times B) \to C \vdash ? : A \to (B \to C)$

Solution:

$$\dfrac{\dfrac{f \ : \ (A \times B) \to C \quad \dfrac{\dfrac{}{a : A}\,1 \quad \dfrac{}{b : B}\,2}{(a, b) : A \times B}\,\times}{\dfrac{f(a, b) \ : \ C}{\lambda y.\ f(a, y) \ : \ B \to C}\,\lambda, 2}\,\mathsf{APP}}{\lambda x.\lambda y.\ f(x, y) \ : \ A \to (B \to C)}\,\lambda, 1$$

Compare this typing derivation to the natural deduction verifying the sequent:

$$(A \wedge B) \to C \ \vdash \ A \to (B \to C)$$

$$\dfrac{\dfrac{\dfrac{(A \wedge B) \to C \quad \dfrac{\dfrac{}{A}\,1 \quad \dfrac{}{B}\,2}{A \wedge B}\,\wedge I}{C}\,\mathsf{MP}}{B \to C}\,\to I, 2}{A \to (B \to C)}\,\to I, 1$$

The resulting λ-term (i.e. program) below

$$\lambda x.\lambda y.\ f(x, y) \ : \ A \to (B \to C)$$

is the proof-object witnessing the proof of the sequent.

(b) $f : A \to (B \to C) \ \vdash\ ? : (A \times B) \to C$

Solution:

$$\dfrac{\dfrac{\dfrac{f : A \to (B \to C) \qquad \dfrac{\overline{p : A \times B}^{\;1}}{\mathsf{FST}\ p \ :\ A}\ \mathsf{FST}}{f\ (\mathsf{FST}\ p)\ :\ B \to C}\ \mathsf{APP} \qquad \dfrac{\overline{p : A \times B}^{\;1}}{\mathsf{SND}\ p :\ B}\ \mathsf{SND}}{(f\ (\mathsf{FST}\ p))\ (\mathsf{SND}\ p)\ :\ C}\ \mathsf{APP}}{\lambda x.\ (f\ (\mathsf{FST}\ x))\ (\mathsf{SND}\ x)\ :\ (A \times B) \to C}\ \lambda, 1$$

Compare this typing derivation to the natural deduction verifying the sequent:

$$A \to (B \to C)\ \vdash\ (A \times B) \to C$$

$$\dfrac{\dfrac{\dfrac{A \to (B \to C) \qquad \dfrac{\overline{A \wedge B}^{\;1}}{A}\ \wedge E_l}{B \to C}\ \mathsf{MP} \qquad \dfrac{\overline{A \wedge B}^{\;1}}{B}\ \wedge E_r}{C}\ \mathsf{MP}}{(A \wedge B) \to C}\ \to I, 1$$

The resulting λ-term (i.e. program) below

$$\lambda x.\ (f\ (\mathsf{FST}\ x))\ (\mathsf{SND}\ x)\ :\ (A \times B) \to C$$

is the proof-object witnessing the proof of the sequent.

(c) $f : A \to B,\ g : B \to C \vdash ?: A \to C$

Solution:

$$\dfrac{\dfrac{g\ :\ B \to C \qquad \dfrac{f : A \to B \quad \overline{a : A}^{\,1}}{f\ a\ :\ B}\,\mathsf{APP}}{g(f\ a)\ :\ C}\,\mathsf{APP}}{\lambda x.\ g(f\ x)\ :\ A \to C}\,\lambda, 1$$

Compare this typing derivation to the natural deduction verifying the sequent:

$$A \to B,\ B \to C \vdash\ A \to C$$

$$\dfrac{\dfrac{B \to C \qquad \dfrac{A \to B \quad \overline{A}^{\,1}}{B}\,\mathsf{MP}}{C}\,\mathsf{MP}}{A \to C}\,\to I, 1$$

The resulting λ-term (i.e. program) below

$$\lambda x.\ g(f\ x)\ :\ A \to C$$

is the proof-object witnessing the proof of the sequent.

(d) $p : A + B,\ f : A \to C,\ g : B \to D \vdash\ ? : C + D$

Solution:

$$\dfrac{p : A + B \qquad \dfrac{\dfrac{\dfrac{f : A \to C \quad \overline{a : A}^{\,1}}{f\ a : C}\,\mathsf{APP}}{\mathsf{inl}\ (f\ a) : C + D}\,\mathsf{inl}}{\lambda x.\ \mathsf{inl}\ (f\ x) : A \to C + D}\,\lambda, 1 \qquad \dfrac{\dfrac{\dfrac{g : B \to D \quad \overline{b : B}^{\,2}}{g\ b : D}\,\mathsf{APP}}{\mathsf{inr}\ (g\ b) : C + D}\,\mathsf{inr}}{\lambda y.\ \mathsf{inl}\ (g\ y) : B \to C + D}\,\lambda, 2}{\mathsf{cases}\ p\ (\lambda x.\ \mathsf{inl}\ (f\ x))\ (\lambda y.\ \mathsf{inr}\ (g\ y))\ :\ C + D}\,\mathsf{cases}$$

Compare this typing derivation to the natural deduction verifying the sequent:

$$A \lor B,\ A \to C,\ B \to D \vdash\ C \lor D$$

$$\dfrac{A \lor B \qquad \dfrac{\dfrac{\dfrac{A \to C \quad \overline{A}^{\,1}}{C}\,\mathsf{MP}}{C \lor D}\,\lor I}{A \to C \lor D}\,\to I, 1 \qquad \dfrac{\dfrac{\dfrac{B \to D \quad \overline{B}^{\,2}}{D}\,\mathsf{MP}}{C \lor D}\,\lor I}{B \to C \lor D}\,\to I, 2}{C \lor D}\,\lor E$$

The resulting λ-term (i.e. program) below

$$\mathsf{cases}\ p\ (\lambda x.\ \mathsf{inl}\ (f\ x))\ (\lambda y.\ \mathsf{inr}\ (g\ y))\ :\ C + D$$

is the proof-object witnessing the proof of the sequent.

Extras: For these extra problems consider $\perp$ to be type with no constructor or destructors. Furthermore, consider $\neg P$ to be shorthand for the function type: $\neg P := P \to \perp$.

(a) $p : \neg A + \neg B \vdash ? : \neg(A \times B)$

(b) $p : \neg(A + B) \vdash ? : \neg A \times \neg B$

(c) $p : \neg A \times \neg B \vdash ? : \neg(A + B)$

(d) $f : A \to C,\ g : B \to D,\ p : \neg C + \neg D \vdash ? : \neg A + \neg B$

(e) $a : A,\ f : \neg A \vdash ? : \neg B$

(f) $f : A \to B,\ g : A \to \neg B \vdash ? : \neg A$

Solutions: Use the natural deductions from Tutorial 2 to help write these programs :)

5. This exercise shows you an example of a general observation first made by William Tait, relating the simplifications of proofs and the process of computation in the λ-calculus.

   Consider the following proof of the theorem

$$\vdash \; A \wedge B \to B$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \wedge B}^{\,1}}{B}\,\wedge E_R \qquad\qquad \dfrac{\overline{A \wedge B}^{\,1}}{A}\,\wedge_L}{B \wedge A}\,\wedge I}{B}\,\wedge E_L}{A \wedge B \to B}\,\to, 1$$

   (a) Determine the corresponding proof-object for this proof.

   (b) Why does the proof-object have a redex in it?

   (c) Perform the β-reduction on the proof object from (a).

   (d) What proof does the reduced proof-object correspond to?

   Solution:

   The natural deduction proof stated in the question corresponds to the following type construction:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{p \;:\; A \times B}^{\,1}}{B}\,\mathsf{SND} \qquad\qquad \dfrac{\overline{p \;:\; A \times B}^{\,1}}{A}\,\mathsf{FST}}{B \times A}\,\times}{B}\,\mathsf{FST}}{\lambda x : A \times B.\ \mathsf{FST}(\mathsf{SND}x, \mathsf{FST}x) \;:\; A \times B \to B}\,\lambda, 1$$

   The corresponding proof-object is

$$\lambda x : A \times B.\ \mathsf{FST}(\mathsf{SND}x, \mathsf{FST}x) \;:\; A \times B \to B$$

   Which can be B-reduced to

$$\lambda x : A \times B.\ \mathsf{SND}x \;:\; A \times B \to B$$

   This proof-object takes in a pair and returns the second of the pair.

   As a natural deduction this corresponds to the following:

$$\dfrac{\dfrac{\overline{A \wedge B}^{\,1}}{B}\,\wedge E_R}{A \wedge B \to B}\,\to, 1$$

   This simplified program corresponds to a shorter proof. In this sense B-reduction (i.e. computation!) is related to the simplification of proofs.

6. Prove that the type $(A \to A) \to A$ is uninhabited i.e. there is no term t of simple type theory that has this type.

$$\nvdash t : (A \to A) \to A$$

Solution

We know $\not\models (A \to A) \to A$ because their is a counterexample. This implies there can be no derivation of this formulae. So, by the Curry-Howard isomorphism, we conclude there can be no term of this type in the simply typed λ-calculus.