MATH230: Tutorial Seven [Solutions]

Recursion and Combinatory Logic

Key ideas
- Write recursive processes in λ-calculus,
- Write higher order procedures in λ-calculus,
- Prove extensional identities in combinatory logic,
- Translate between λ-calculus and combinatory logic.

Relevant topic: Untyped Lambda Calculus Slides
Relevant reading: Type Theory and Functional Programming, Simon Thompson

Hand in exercises: 1b, 4c, 5c, 6a, 7c
Due Friday @ 5pm to the submission box on Learn.

Discussion Questions

- 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.

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

  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:

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

  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:

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

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

   (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:

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

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

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

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

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

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

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

Solution:

Recall the definition of the Fibonacci sequence:

$$\begin{aligned}\text{Fib } 0 &= 0\\ \text{Fib } 1 &= 1\\ \text{Fib } n &= \text{Fib}(n-1) + \text{Fib}(n-2)\end{aligned}$$

As this recursive definition has two base cases, we will need nested COND to check for each of these. In the third condition we will need to recursively call the helper function twice; once for each summand.

$$\begin{aligned}\text{GO} :\equiv\ \lambda s.\ \lambda n.\ &\text{COND } (\text{ZERO? } n)\\ &\quad \text{ZERO}\\ &\quad (\text{COND } (\text{ZERO? } (\text{PRED } n))\\ &\qquad \text{ONE}\\ &\qquad (\text{SUM } (s\ (\text{PRED } n))\\ &\qquad\quad (s\ (\text{PRED } (\text{PRED } n)))))\end{aligned}$$

Indentation is used to aid readability. COND has three inputs; these are all aligned on new lines. SUM has two inputs, these are aligned on new lines.

We use the helper function GO to define

$$\text{FIB} := \text{Y GO}$$

Notice that once we have written a few fundamental λ-terms, a lot of the programming now happens at a level higher than pure lambdas and feels closer to actual programming. Nonetheless, all of these terms with their syntactic sugar can be desugared down to their definitions and computed using just β-reduction.

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.

   Solution: We will write a procedure that computes the remainder when n is divided by d. It will do the naive thing of subtracting d from n until $n < d$.

$$\begin{aligned}\mathsf{GO} :\equiv \lambda s.\ \lambda n.\ \lambda d.\ \mathsf{COND}\ &(>?d\ n)\\ &n\\ &(s\ (-\ n\ d)\ d)\end{aligned}$$

$$\mathsf{REM} := \mathsf{Y}\ \mathsf{GO}$$

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

   In order to determine whether d divides n we need only check the remainder when n is divided by d.

$$\mathsf{DIVIDES?} :\equiv \lambda n.\ \lambda d.\ \mathsf{ZERO?}\ (\mathsf{REM}\ n\ d)$$

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

   Solution: We will write a procedure $\overline{\mu}$ which takes in a predicate, p, a lower bound l, and an upper bound u. It returns the smallest solution to p in the interval $[l, u]$. If there are no such solutions, then it returns FALSE.

$$\begin{aligned}\mathsf{GO} :\equiv \lambda s.\ \lambda p.\ \lambda l.\ \lambda u.\ \mathsf{COND}\ &(>?\ l\ u)\\ &\mathsf{FALSE}\\ &(\mathsf{COND}\ (p\ l)\\ &\qquad p\\ &\qquad (s\ p\ (\mathsf{SUCC}\ l)\ u))\end{aligned}$$

   Using this helper function we define the bounded search operator $\overline{\mu}$

$$\overline{\mu} := \mathsf{Y}\ \mathsf{GO}$$

   Note: the helper can be adjusted to return TRUE instead of the specific solution.

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

   In order to determine whether n is prime we need only search for a divisor from 2 upto n-1. This a bounded search for a solution to the predicate (DIVIDES? n). We have used partial application of DIVIDES? to build a unary predicate that takes in a single number and determines whether it divides n. If there is a divisor found in the bounded search, then the number is composite.

$$\mathsf{COMPOSITE?} :\equiv\ \lambda x.\ \overline{\mu}\ (\mathsf{DIVIDES?}\ x)\ \mathsf{TWO}\ (-\ x\ \mathsf{ONE})$$

$$\mathsf{PRIME?} :\equiv\ \lambda x.\ \mathsf{NOT}\ (\mathsf{COMPOSITE?}\ x)$$

   This all assumes the input is greater than 2. How can we fix this bug?

4. In lectures we introduced a λ-term for computing the sum of a sequence of consecutive integers. This used the helper-function:

$$\begin{aligned}\mathsf{GO} :\equiv \lambda s.\ \lambda a.\ \lambda l.\ \lambda u.\ \mathsf{COND}\ (>?\ l\ u)\\ &\quad a\\ &\quad (s\ (\mathsf{SUM}\ a\ l)\ (\mathsf{SUCC}\ l)\ u)\end{aligned}$$

We defined ACCUMULATE = Y GO. Make alterations to the helper-function to compute the following:

(a) Compute the sum of the squares of each integer, $\sum_{i=l}^{u} i^2$ **Solution:**

$$\begin{aligned}\mathsf{GO} :\equiv \lambda s.\ \lambda a.\ \lambda l.\ \lambda u.\ \mathsf{COND}\ (>?\ l\ u)\\ &\quad a\\ &\quad (s\ (\mathsf{SUM}\ a\\ &\qquad (\mathsf{MULT}\ l\ l))\\ &\quad (\mathsf{SUCC}\ l)\\ &\quad u)\end{aligned}$$

(b) Compute the sum of each term passed through an arbitrary function, $\sum_{i=l}^{u} f(i)$ **Solution:**

$$\begin{aligned}\mathsf{GO} :\equiv \lambda s.\ \lambda a.\ \lambda l.\ \lambda u.\ \mathsf{COND}\ (>?\ l\ u)\\ &\quad a\\ &\quad (s\ (\mathsf{SUM}\ a\\ &\qquad (f\ l))\\ &\quad (\mathsf{SUCC}\ l)\\ &\quad u)\end{aligned}$$

Note: the function f could be factored out into the abstractions.

(c) Compute the sum of those terms in the interval that satisfy some predicate $P?(x)$. **Solution:**

$$\begin{aligned}\mathsf{GO} :\equiv \lambda s.\ \lambda a.\ \lambda l.\ \lambda u.\ \mathsf{COND}\ (>?\ l\ u)\\ &\quad a\\ &\quad (\mathsf{COND}\ (P?\ l)\\ &\qquad (s\ (\mathsf{SUM}\ a\ l)\\ &\qquad\quad (\mathsf{SUCC}\ l)\\ &\qquad\quad u)\\ &\qquad (s\ a\\ &\qquad\quad (\mathsf{SUCC}\ l)\\ &\qquad\quad u))\end{aligned}$$

Note: the predicate P? could be factored out into the abstractions.

PTO

Recall the following reduction rules of the CL combinators.

$\mathsf{S}xyz \to_\beta xz(yz)$ $\quad$ $\mathsf{K}xy \to_\beta x$
$\mathsf{I}x \to_\beta x$ $\quad$ $\mathsf{B}fgx \to_\beta f(gx)$
$\mathsf{W}fx \to_\beta fxx$

5. Verify each of the following extensional equality claims by evaluating each side at an appropriate number of variables and check the reductions are identical.

   (a) I = SKK

   (b) SK = KI

   (c) B = S(KS)K

   (d) W = SS(KI)

6. Each of these CL terms are reducible. If they have a normal form, then compute it. Otherwise, show that the term has no normal form.

   (a) SKI(KIS)

   (b) KS(I(SKSI))

   (c) SKIK

   (d) SII(SII)

7. Translate each of these λ-terms into combinatory logic expressions involving only SKI combinators (and free variables) using the translation defined in the lecture slides.

   (a) $\lambda x.\ \lambda y.\ y$

   (b) $\lambda x.\ x\ x$

   (c) $(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

   (d) $\lambda u.\ \lambda v.\ u\ v$

   (e) $\lambda x.\ f\ (x\ x)$

   (f) $\lambda f.\ \mathsf{S}(\mathsf{K}f)(\mathsf{SII})$

   (g) $\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$