Programming Paradigms: Exercises

December 5, 2012

1 Generalities

1.1 Paradigms, Languages, Features

Consider the language C++ (or your favourite programming language, ...).

Exercise 1 Write a list of features (programming constructs) implemented in C++. Be as exhaustive as you can (list at least 10 features). [*]

Exercise 2 For each programming paradigm (Imperative, OO, etc.), evaluate how well C++ supports that paradigm. Argue using the list compiled in the previous answer. [*]

Exercise 3 Can you identify a paradigm not studied in the course which is supported by C++? [***]

1.2 Types

Exercise 4 Give a meaningful type to the following values. [*]

1. 4
2. 123.53
3. 1/3
4. π
5. 'c'
6. "Hello, world!"
7. -3
8. (unary) -
9. (binary) +

10. sin
11. derivative

Exercise 5 Explain the meaning of the following types. (Hint: what kind of values can inhabit those types?) [**]

1. String
2. String → String
3. String → String → String
4. (String → String) → String
5. String → (String → String)
6. (String → String) → (String → String)

One can not only parameterize over values, but also over types. (Eg. in Java, generic classes).

For example, the following type is a suitable type for a sorting function: it expresses that the function works for any element type, as long as you can compare its inhabitants.

∀ a. (a → a → Bool) → Array a → Array a

Exercise 6 Does sort in your favourite language have a similar type? How close/far is it? [**]

Consider the type

$$\forall a\, b.\mathsf{Pair}\, a\, b \to \mathsf{Pair}\, b\, a$$

Exercise 7 What can possibly inhabit it? [***]

2 Imperative Programming

2.1 Gotos to loops

Consider the algorithm:

1. **Preliminary:** A is an array of integers of length m and B is an array of integers of length n. Also the elements from both arrays are distinct (from the elements in both arrays) and in ascending order.
2. **Step1:** if n or m is zero **STOP**. Otherwise if $m>n$, set $t := \lfloor log\,(m/n) \rfloor$ and go to **Step4**, else set $t := \lfloor log\,(n/m) \rfloor$.
3. **Step2:** compare *A[m]* with *B[n + 1 - 2^t]*. If *A[m]* is smaller, set $n := n - 2^t$ and return to **Step1**.
4. **Step3:** using binary search(which requires exactly t more comparisons), insert *A[m]* into its proper place among *B[n + 1 -2^t] ... B[n]*. If k is maximal such that *B[k] <A[m]*, set *m := m - 1* and $n := k$. Return to **Step1**.
5. **Step4:**(Step 4 and 5 are like 2 and 4, interchanging the roles of n and m, A and B) if *B[n] <A[m+1-2^t]*, set $m := m - 2^t$ and return to **Step1**.
6. **Step5:** insert *B[n]* into its proper place among the As. If k is maximal such that *A[k] <B[n]*, set $m := k$ and *n := n - 1*. Return to **Step1**.

Exercise 8 implement binary search without gotos in the context of the algorithm. There is a slight change compared to the classical algorithm, since the scope of the search is different. Keep in mind! [1] [*]

Exercise 9 **Step1** may require the calculation of the expression $\lfloor log\,(m/n) \rfloor$, for $n \geq m$. Explain how to compute this easily without division or calculation of a logarithm. [-]

Exercise 10 Why does the binary search mentioned in the algorithm always take t steps ? [*]

Exercise 11 Explain the behaviour of the algorithm for the arrays $A = \{87, 503, 512\}$ and $B = \{61, 154, 170, 275, 426, 509, 612, 653, 677, 703, 765, 897, 908\}$. [*]

Exercise 12 Implement the above-mentioned algorithm without using *gotos* in a programming language of your choice. Check your implementation with the A and B from the previous question. [**]

[1] If that is too easy, do it for red-black trees.

2.2 Control flow statements to gotos

Exercise 13 Translate the following **for** loop with explicit gotos: [*]

for (statement1; condition; statement2)
loop_body

Exercise 14 Translate the following **foreach** loop with explicit gotos: [*]

```
foreach i in k..l do
  body
```

Exercise 15 Translate the do/while construct. [*]

Exercise 16 Translate the switch/case construct. [*]
(If you want to make sure your translation is correct you should check the specification of the C language. `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf`)

Exercise 17 Translate the insertion sort algorithm. [*]

2.3 Pointers and call by reference

Exercise 18 Create a binary search tree where nodes contain integer number in C/C++/Java. Implement a function that adds a value to the tree and one that deletes a certain value from the tree. [*]

Exercise 19 Write a recursive traversal of the above tree. [*]

Exercise 20 Write a swap function using call by reference. [*]

Exercise 21 Write a swap function using call by value. [*]

Exercise 22 Does Java use call by reference? Give examples to support your answer. [**]

Exercise 23 Write down pros and cons of using call by reference vs. call by value. (Ease of use, performance, ...) [**]

2.4 More on control flow and pointers: Duff's device

Duff's device is an optimization idiom for serial copies in the language C. The fragment below lists first the original code, then Duff's optimization.

```
/* (Almost) original code */
int main() {
   short *to, *from;
   int count;
   ...
   {
     /* pre: count > 0 */
     do
          *to++ = *from++;
     while(--count>0);
   }
   return 0;
}
```

Many things happen in the assignment "*to++ = *from++;". Can you figure out what exactly?

Exercise 24 Translate the above to multiple statements, so that for each of them only one variable (or memory location) is updated. Explain in your own words what happens (draw a picture if necessary). [*]

Duff optimised the above code as follows:

```
/* Duff's transformation */
int main() {
  short *to, *from;
  int count;
  ...
  {
     /* pre: count > 0 */
     int n = (count + 7) / 8;
     switch(count % 8){
       case 0:  do{     *to++ = *from++;
       case 7:           *to++ = *from++;
       case 6:           *to++ = *from++;
       case 5:           *to++ = *from++;
       case 4:           *to++ = *from++;
       case 3:           *to++ = *from++;
       case 2:           *to++ = *from++;
       case 1:           *to++ = *from++;
                } while (--n > 0);
     }
  }
  return 0;
}
```

Exercise 25 Translate the switch statements to gotos. [**]

Is the second program really equivalent to the first?

Exercise 26 Show that the instruction "`*to++ = *from++`" will be executed `count` times in the second program. [-]

Exercise 27 Explain the equivalence by a series of program transformations. [****]

Exercise 28 Can you guess why Duff expects the second program to be faster? What instructions are executed in the first program but not in the second? [-]

Further reading

- For the original email in which Tom Duff presented his "device", see `http://www.lysator.liu.se/c/duffs-device.html`
- You can see the assembly code generated by gcc by compiling with

gcc -S <filename>.

2.5 From recursion to explicit stack

Exercise 29 Re-implement the tree traversal (from above), but using explicit stacks. [**]

Consider the following recursive equation for computing Fibonacci numbers:

$$fib_{n+2} = fib_{n+1} + fib_n$$

Exercise 30 Implement the recursive function computing the n-th Fibonacci number based on the expression above. [*]

Exercise 31 Why is it not efficient? [*]

Exercise 32 Implement a slightly-optimized recursive function for the same purpose. (Hint: use an accumulator parameter). [*]

Exercise 33 Implement a version of each of the two functions above by using an explicit stack. [**]

Exercise 34 Implement the Ackermann function without recursion. (See `http://en.wikipedia.org/wiki/Ackermann_function`). [**]

Exercise 35 Implement the algorithm from the previous section without loops (only recursion allowed). [**]

Exercise 36 Translate the quicksort algorithm. [**]

Exercise 37 Implement the following algorithm as a recursive function. (And remove the loop!) [**]

```
a = 0
b = 1
for i in [1..n] do
  c = a + b
  a = b
  b = c
return a
```

3 Object-Oriented Programming

Consider the following code, in C# syntax:

```
interface Monoid {
   Monoid op(Monoid second);
   Monoid id();
};

struct IntegerAdditiveMonoid : Monoid {
    public IntegerAdditiveMonoid(int x) {
        elt = x;
    }
    public IntegerAdditiveMonoid op(IntegerAdditiveMonoid second) {
        return new IntegerAdditiveMonoid(
            elt + second.elt);
    }
    public IntegerAdditiveMonoid id(){
        return new IntegerAdditiveMonoid(0);
    }
    int elt;
};
```

3.1 Explicit method pointers

Exercise 38 Translate the above code to a C-like language, using explicit method pointers. (Hint: you can simply consider the interface as a class without fields.) [*]

Exercise 39 Briefly recap: what is a *monoid*, mathematically? [-]

Exercise 40 Give two examples of data types that can be considered monoids. (Hint: Strings would form a monoid under the appropriate structure; what is the structure? [-]

Exercise 41 Write another instance of the monoid interface, using one of the examples you found. Also write its translation to a C-like language. [*]

Exercise 42 Assume variables a,b of type Monoid. Translate the expression a.op(b). [*]

Exercise 43 Assume to objects x,y of two different instances of Monoid are bound to the variables a,b. Explain what happens at runtime when the expression is evaluated. (Which code is executed?) [*]

3.2 Co/Contra variance

Surprise: the above code is refused by the C# compiler:

```
> gmcs Monoids.cs
Monoids.cs(6,8): error CS0535: 'IntegerAdditiveMonoid' does not implement
  interface member 'Monoid.op(Monoid)'
Monoids.cs(2,11): (Location of the symbol related to previous error)
Monoids.cs(6,8): error CS0535: 'IntegerAdditiveMonoid' does not implement
  interface member 'Monoid.id()'
Monoids.cs(3,11): (Location of the symbol related to previous error)
Compilation failed: 2 error(s), 0 warnings
```

Exercise 44 What if the method op *would* compile? Define objects a,b, of appropriate types, so that a.op(b), if is run, would result in a run-time error. [*]

Exercise 45 What if the method id would compile? Could you construct a similar run-time error? (Hint: do the translation job if the answer is not obvious to you.) [*]

Exercise 46 Explain the error messages in terms of co-/contra-/nonvariance. [**]

Exercise 47 The corresponding program in Java behaves differently. Briefly explain how, consult the Java Language Specification (JLS), and back your answer with the appropriate clause(s) in the JLS. [***]

Exercise 48 Can you change the code so that the (current) C#-compiler accepts it? What is the problem then? [-]

4 Functional Programming

4.1 Algebraic Data Types and Pattern Matching

Exercise 49 Define an algebraic type for binary trees **[*]**

Exercise 50 define an algebraic type for arithmetic expressions **[*,@3]**

Exercise 51 define a simple interpreter for the above type **[*,@3]**

Exercise 52 Translate the above 2 structures to an OO language. (Hint: One class corresponds to leaves, one to branching.) **[*,@3]**

Exercise 53 Translate the interpreter. You are not allowed to use 'instanceof' **[*,@3]**

Exercise 54 Translate the interpreter. You must use 'instanceof'. **[*]**

4.2 Currification and partial application

Exercise 55 Define a function f following this spec: given a integer, return it unchanged if it is greater than zero, and zero otherwise. (The type must be Int $\rightarrow$ Int.) **[*,@3]**

Exercise 56 Assuming a function max : (Int $\times$ Int) $\rightarrow$ Int, define the function f. **[*,@3]**

Exercise 57 Define a function max', by currifying the function max. **[*]**

Exercise 58 Define f using max'. **[*]**

4.3 Higher-order functions

Assume the filter, map, foldr functions as in the Haskell prelude. f comes from the previous section.

Exercise 59 Unfold the following expressions: **[*,@3]**

1. map f
2. filter (>= 0)
3. foldr [] (++)

Consider the following imperative program:

```
for (i=0;i<sizeof(a);i++)
  if (a[i].grade >= 24)
     *b++ = a[i];
```

Exercise 60 How would the same algorithm be naturally expressed in a functional language? (Use functions from the Haskell prelude to shorten your code) [*,@3]

Exercise 61 write a function that does the dot-product of two vectors; [*]

Exercise 62 make an abstract version of the above. [*]

Exercise 63 Can you find the function you created in the Haskell Data.List module? [**]

Exercise 64 The standard function `insert` inserts an element into an ordered list, for example []

```
insert 4 [1,3,5,7] = [1,3,4,5,7]
```

Define a function

```
sort :: [Integer] -> [Integer]
```

to sort lists, using `insert`.

Exercise 65 Express `sort` in terms of `foldr` (do not use explicit recursion). []

4.4 Closures

Consider the program:

```
test1 xs = foldr (+) 0 xs
test2 xs = foldr (*) 1 xs
```

Exercise 66 Identify higher-order applications in the above program. [*,@4]

Exercise 67 Assuming that `xs` are lists of integers, replace the above uses of higher-order application by explicit closures. (Hint: you need to also change the definition of foldr/map). [*,@4]

Exercise 68 Add the following line to the above program and repeat the above 2 exercises. [**,@4]

```
replace a b xs = map (\x -> if x == a then b else x) xs
```

Exercise 69 Eratosthenes' sieve is a method for finding prime numbers, dating from ancient Greece. We start by writing down all the numbers from 2 up to some limit, such as 1000. Then we repeatedly do the following: **[*]**

- The *first* number in the list is a prime. We generate it as an output.
- We *remove all multiples* of the first number from the list—including that number itself.
- Loop.

We terminate when no numbers remain in our list. At this point, all prime numbers up to the limit have been found.

The algorithm can be implemented in Haskell as follows:

Write the algorithm in Haskell.

Exercise 70 Consider the following (totally unrelated) algorithm: **[*,@4]**

```
primes = sieve [2..1000]
sieve (n:ns) = n:sieve (filter (not . (`isDivisibleBy` n)) ns)
x `isDivisibleBy` y = x `mod` y == 0
```

Inline the call of function composition in the above as a lambda abstraction. The only remaining higher-order function is filter.

Exercise 71 Write a version of filter which takes an explicit closure as an argument. Remember to write (and use) an apply function. Hint: in order to test your program you need to define an example parameter, such as (>= 0), etc. **[*,@4]**

Exercise 72 Re-write sieve using the above. **[*]**

Exercise 73 Write a version of sieve in imperative (or OO) language, by translation of the answer to the previous exercise. (eg. you can not use arrays instead of lists). (Hint: write this one as a recursive program). **[**]**

4.5 Explicit state

Consider a binary tree structure with integer values in the nodes.

Exercise 74 In the Java version, write a function that replaces each element with its index in preorder traversal of the tree. **[*,@4]**

Exercise 75 Translate the function above to Haskell thinking of it as an imperative algorithm. You should use *IP* from the lecture notes. What is the "state of the world" in this case? **[**,@4]**

Exercise 76 Rewrite the Haskell version, in such a way that passing "state of the world" is made visible as such, that is, eliminate your usage of *IP*. **[*,@4]**

4.6 Laziness

There are no lazy languages that permit mutation.

Exercise 77 Why not? **[***, @5]**

4.6.1 Lazy Dynamic Programming

Consider a the function computing the fibonacci sequence:

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Exercise 78 Estimate the time-complexity of computing $\mathsf{fib}\, n$. **[-]**

One can make the above computation take linear time by saving the intermediate results in an array:

```
fibs[0] = 0
fibs[1] = 1
for i = 2 to n do
  fibs[n] = fibs[n-1] + fibs[n-2]
```

Exercise 79 Instead of specifying the order of filling the array via an imperative algorithm, let Haskell's lazy evaluation take care of it. Write the definition of the array fibs in Haskell. **[*]**

Exercise 80 What portion of the array is forced when $\mathsf{fibs}!k$ is accessed? Draw the graph of computation dependencies for $k = 5$. **[*]**

Exercise 81 Write the Floyd-Warshall algorithm in your favourite imperative language. Assume that there is no weight on edges; and you're only interested in whether there is a path between two given nodes, not the number of steps in the path. **[*, @5]**

Note that the best-case performance is Cubic.

Exercise 82 Repeat the above, but make sure you never overwrite a cell in a matrix. (What do you need to do to make this at all possible?) **[*]**

Exercise 83 Using the same formula, fill the Roy-Warshall matrix using an array comprehension in a lazy language (optionally use explicit thunks for the contents of each cell). Discuss the best-case performance. **[*, @5]**

Exercise 84 Does your favourite spreadsheet program have strict, or lazy logical disjunction operator? Test it. [-]

Exercise 85 Can you write the Roy-Warshall algorithm in it? [-]

Exercise 86 Repeat the above steps with the algorithm to compute the least edit distance. `http://en.wikipedia.org/wiki/Levenshtein_distance` [**]

```
int LevenshteinDistance(char s[1..m], char t[1..n])
{
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t;
  // note that d has (m+1)x(n+1) values
  declare int d[0..m, 0..n]

  for i from 0 to m
    d[i, 0] := i // the distance of any first string to an empty second string
  for j from 0 to n
    d[0, j] := j // the distance of any second string to an empty first string

  for j from 1 to n
  {
    for i from 1 to m
    {
      if s[i] = t[j] then
        d[i, j] := d[i-1, j-1]       // no operation required
      else
        d[i, j] := minimum
                   (
                     d[i-1, j] + 1,  // a deletion
                     d[i, j-1] + 1,  // an insertion
                     d[i-1, j-1] + 1 // a substitution
                   )
    }
  }

  return d[m,n]
}
```

4.6.2 Lazy Lists (Streams)

Remember the sieve algorithm. Lazy lists can be (potentially) infinite, although of course no program can evaluate *all* the elements of an infinite list. Nevertheless, using lazy lists can help us avoid building unnecessary limitations into our code. Check that the `sieve` function also works given the *infinite* list `[2..]`—the output should be the *infinite* list of all prime numbers.

Exercise 87 How would you find the first 100 prime numbers? **[*, @5]**

Remark: start by answering the following questions on a simpler algorithm, for example the function enumFrom which generates the infinite list of numbers starting at a given one.

Exercise 88 Translate sieve to use explicit thunks (as in the lecture) **[*, @5]**

Oh noes, this introduced higher-order function(s).

Exercise 89 Where are the new higher-order functions? **[**]**

You know what's coming...

Exercise 90 Remove higher-orderness using the best technique available. **[***]**

Exercise 91 Write a version of lazy sieve in imperative (or OO) language. **[**, @5]**

Since there are no more functions in the data, you can now display your infinite lists.

Exercise 92 Do so for a few meaningful inputs, and interpret what you see. **[***]**

5 Concurrent Programming

5.1 Channels and Processes

Exercise 93 Write a process that manages a bank account. It has a channel for queries; which can be deposits and withdrawals. The answer is posted to a channel that comes with the query. (Note: you cannot use references to store any state – see variable-managing process for inspiration) **[*, @5]**

Exercise 94 Write a client for the above process that move "money" from an account to another. ("transaction") **[*]**

Exercise 95 Assume that withdrawals/deposits can fail. (For example if there is too much/litte on the account). Modify the server process accordingly. **[*]**

Exercise 96 Is the total amount of money a constant of your system? (Consider that transactions may fail in the "middle".) How can you ensure that it is? Write the corresponding code. **[**, @5]**

Exercise 97 Implement a process that manages a semaphore. The process should implement the requests P and V. (See the wikipedia article for explanation of semaphore, P and V). `http://en.wikipedia.org/wiki/Semaphore_(programming)` [*]

Exercise 98 Implement two library functions that help communicate with the above server. (Hint: you may have to create channels.) [*]

5.2 Explicit continuations

Consider the following outline for the "business logic" of a web-application:

```
session connection = do
   items <- webForm connection "What do you want to buy?"
   address <- webForm connection "Where do you want your stuff delivered?"
   daMoney <- webForm connection "Enter your credit card details, now."
   secureInDatabase daMoney (priceOf items)
   placeOrder items address
```

Exercise 99 What is the purpose, and type of the webForm primitive? [*, @6]

Exercise 100 Transform the type of webForm to take a continuation. [**, @6]

Exercise 101 Break the above code into continuations, linked by the webForm primitive. [**, @6]

Exercise 102 Outline what the webForm function should do. Discuss in particular what happens if the user presses the "back" button in their browser. [***, @6]

Recursion and continuations Remember your interpreter for arithmetic expressions. It should have type:

Expr → Int

Let's make continuations explicit. In this case, the result is not returned directly, but applied to a continuation. Hence, the type becomes:

Expr → (Int → a) → a

Exercise 103 Write a wrapper for the interpreter which has the above type [*, @6]

Exercise 104 Replace every recursive call in the interpreter by a call to the wrapper. (Hint: you must decide what is the "continuation" for every call, and for this you must choose an order of evaluation!) **[***, @6]**

Exercise 105 Unfold the usage of the interpreter in the wrapper. **[***, @6]**

6 Logic Programming

In this section, exercises are sometimes formulated both in Prolog and Curry syntax; as indicated in the margin.

6.1 Metavariables and unification

Exercise 106 What is the result of each of the following unifications? Try to come up with the result without using the prolog/curry interpreter! **[*, @6]**

Prolog

---

a(X,Y) = a(b,c)
a(X,Y) = a(Z,Z)
a(X,X) = a(b,c)
e(X) = a(b,b)
d(X,X,Y) = d(Z,W,W)
a(X,X) = d(Z,W,W)

Curry

```
data X = A X X | B | C | D X X X | E X

A x y =:= A B C  where x, y free
A x y =:= A z z  where x, y, z free
A x x =:= A B C where x free
E x =:= A B B where x free
D x x y =:= D z w w where x, y, w, z free
A x x =:= D z w w where x, y, w, z free
```

Exercise 107 Assume a relation plus relating two natural numbers and their sum. **[*]**

Define a relation minus, relating two natural numbers and their difference, in terms of plus.

6.1.1 Difference Lists

A difference list is a special structure that can support efficient concatenation. It uses unification in a clever way to that end.

The difference-list representation for a list can be obtained as follows:

Prolog

```
fromList ([], d(X,X)).
fromList([A|As],d(A:Out,In)) :- fromList(As,d(Out,In)).
```

Curry

```
data DList a = D [a] [a]

fromList :: [a] -> D a -> Success
fromList [] (D x x') = x =:= x'
fromList (a:as) (D (a':o) i) = a =:= a' & fromList as (D o i)
```

A structure of the form d(Out,In) will represent the list L if Out unifies with L concatenated with In. Or, less technically, a list L will be represented as the difference between Out and In: so for instance,

Prolog

$$[1,2] \longrightarrow d([1,2,3,4],[3,4])$$

Curry

$$[1,2] \longrightarrow D[1,2,3,4][3,4]$$

You can check how fromList works by testing it: Prolog

```
fromList ([1,2,3], X).
```

Curry

```
fromList [1,2,3] x where x free
```

Note that the same metavariable (G271 in my implementation) is present twice in the result. Note that we can get the original result back by unifying this metavariable with the empty list.

Exercise 108 Write a predicate toList :: DList a → [a] → Success to get back to the normal list representation. **[**, @6]**

Given that representation for lists, it is possible to perform concatenation by doing unification only!

Exercise 109 Write a predicate dconcat to concatenate two difference lists, without using the direct representation. **[**, @6]**

Prolog

```
dconcat(X,Y,Z), fromList ([1,2,3], X), fromList ([4,5], Y), toList(Z, Final).
```

Curry

```
dconcat x y z & fromList [1,2,3] x & fromList [4,5] y & toList z final where x, y, z, final
```

should produce:

```
...
final = [1,2,3,4,5]
```

Exercise 110 What happens when you concatenate a difference list with itself? [***]

6.2 Functions ↔ Relations

Consider the following haskell function, that splits a list of integers into two lists: one containing the positive ones (and zero), the other containing the negative ones.

```
split [] = ([],[])
split (x:xs) | x >= 0 = (x:p,n)
             | x <  0 = (p,x:n)
  where (p,n) = split xs
```

If written as a predicate, it is natural if it takes 3 arguments. For example,

```
split([3,4,-5,-1,0,4,-9],p,n)
```

should bind:

```
P = [3,4,0,4]
N = [-5,-1,-9].
```

Exercise 111 Write the predicate split. Make sure that it is *reversible*. That is, you can recover the original list if you pass it the split lists. [**]

Exercise 112 What are the lists that are returned by split/3 when used in reverse? Can it fail? [**]

6.3 Explicit Search

Consider the following list comprehension, in Haskell syntax:

```
c = [f x | z <- a, y <- g z, x <- h y, p v]
```

Exercise 113 Write down possible types for f, a, g, h and p. [*]

Exercise 114 Assume that the above functions/values (f, a, g, h and p) are translated to relational style. What would be natural types for them? [*]

Exercise 115 Translate the list comprehension to relational style. [*]

Exercise 116 Translate all the functions from Family.curry in the "list of successes" style, for all directions. [**]