Programming Languages
functional part 7

2020

Instructor: Odelia Schwartz

Haskell versus ML

Pattern Matching

Haskell:

```
fact 0 = 1
fact 1 = 1
fact n = n * fact(n-1)
```

ML:

```
fun fact(0) = 1
  | fact(1) = 1
  | fact(n:int) = n*fact(n-1);
```

Haskell

Pattern Matching

Could also do with guards and otherwise:

```
fact n
   | n==0 = 1
   | n==1 = 1
   | otherwise = n*fact(n-1)

main :: IO ()
main =  do
  print(fact(4))
```

Cons, car, cdr comparison

Haskell:

```
append ([],lis2) = lis2
append(h:t,lis2) = h:append(t,lis2)
```

ML:

```
fun append ([],lis2) = lis2
| append(h::t,lis2) =
h::append(t,lis2)
```

Scheme:

```
(define (append lis1 lis2)
(cond
 ((null? lis1) lis2)
   (else (cons (car lis1)
     (append (cdr lis1) lis2)))
))
```

Haskell

What kind of sort is this?

```
sort [] = []
sort (h:t) =
        sort [b | b <- t, b <= h]
        ++ [h] ++
        sort [b | b <- t, b > h]

main :: IO ()
main =  do
        print([1, 2] ++ [3,4])
        print(sort [25, 1, 3])
        print(sort [9, 6, 25, 1, 3])
```

Haskell

Compare to some imperative languages!

```pascal
 * A Pascal quicksort.
 *****************************************************************************}
PROGRAM Sort(input, output);
    CONST
        { Max array size. }
        MaxElts = 50;
    TYPE
        { Type of the element array. }
        IntArrType = ARRAY [1..MaxElts] OF Integer;

    VAR
        { Indexes, exchange temp, array size. }
        i, j, tmp, size: integer;

        { Array of ints }
        arr: IntArrType;

    { Read in the integers. }
    PROCEDURE ReadArr(VAR size: Integer; VAR a: IntArrType);
        BEGIN
            size := 1;
            WHILE NOT eof DO BEGIN
                readln(a[size]);
                IF NOT eof THEN
                    size := size + 1
            END
        END;

           PROCEDURE QuicksortRecur(start, stop: integer);
            VAR
                m: integer;

                { The location separating the high and low parts. }
                splitpt: integer;

            { The quicksort split algorithm.  Takes the range, and
              returns the split point. }
            FUNCTION Split(start, stop: integer): integer;
                VAR
                    left, right: integer;       { Scan pointers. }
                    pivot: integer;             { Pivot value. }

                { Interchange the parameters. }
                PROCEDURE swap(VAR a, b: integer);
                    VAR
                        t: integer;
                    BEGIN
                        t := a;
                        a := b;
                        b := t
                    END;

                BEGIN { Split }
                    { Set up the pointers for the hight and low sections, and
                      get the pivot value. }
                    pivot := arr[start];
                    left := start + 1;
                    right := stop;

                    { Look for pairs out of place and swap 'em. }
                    WHILE left <= right DO BEGIN
                        WHILE (left <= stop) AND (arr[left] < pivot) DO
                            left := left + 1;
                        WHILE (right > start) AND (arr[right] >= pivot) DO
                            right := right - 1;
                        IF left < right THEN
                            swap(arr[left], arr[right]);
                    END;

                    { Put the pivot between the halves. }
                    swap(arr[start], arr[right]);

                    { This is how you return function values in pascal.
                      Yeccch. }
                    Split := right
                END;

            BEGIN { QuicksortRecur }
                { If there's anything to do... }
                IF start < stop THEN BEGIN
                    splitpt := Split(start, stop);
                    QuicksortRecur(start, splitpt-1);
                    QuicksortRecur(splitpt+1, stop);
                END
            END;

        BEGIN { Quicksort }
            QuicksortRecur(1, size)
        END;

    BEGIN
        { Read }
        ReadArr(size, arr);

        { Sort the contents. }
        Quicksort(size, arr);

        { Print. }
        FOR i := 1 TO size DO
            writeln(arr[i])
    END.
```

Source: http://sandbox.mc.edu/~bennet/cs404/doc/qsort_pas.html

```java
/*************************************************************************
 *  Quicksort code from Sedgewick 7.1, 7.2.
 *************************************************************************/
 public static void quicksort(double[] a) {
     shuffle(a);                            // to guard against worst-case
     quicksort(a, 0, a.length - 1);
 }

 // quicksort a[left] to a[right]
 public static void quicksort(double[] a, int left, int right) {
     if (right <= left) return;
     int i = partition(a, left, right);
     quicksort(a, left, i-1);
     quicksort(a, i+1, right);
 }

 // partition a[left] to a[right], assumes left < right
 private static int partition(double[] a, int left, int right) {
     int i = left - 1;
     int j = right;
     while (true) {
         while (less(a[++i], a[right]))      // find item on left to swap
             ;                               // a[right] acts as sentinel
         while (less(a[right], a[--j]))      // find item on right to swap
             if (j == left) break;           // don't go out-of-bounds
         if (i >= j) break;                  // check if pointers cross
         exch(a, i, j);                      // swap two elements into place
     }
     exch(a, i, right);                      // swap with partition element
     return i;
 }

 // is x < y ?
 private static boolean less(double x, double y) {
     comparisons++;
     return (x < y);
 }

 // exchange a[i] and a[j]
 private static void exch(double[] a, int i, int j) {
     exchanges++;
     double swap = a[i];
     a[i] = a[j];
     a[j] = swap;
 }

 // shuffle the array a[]
 private static void shuffle(double[] a) {
     int N = a.length;
     for (int i = 0; i < N; i++) {
         int r = i + (int) (Math.random() * (N-i));   // between i and N-1
         exch(a, i, r);
     }
 }



 // test client
 public static void main(String[] args) {
     int N = Integer.parseInt(args[0]);

     // generate N random real numbers between 0 and 1
     long start = System.currentTimeMillis();
     double[] a = new double[N];
     for (int i = 0; i < N; i++)
         a[i] = Math.random();
     long stop = System.currentTimeMillis();
     double elapsed = (stop - start) / 1000.0;
     System.out.println("Generating input:  " + elapsed + " seconds");

     // sort them
     start = System.currentTimeMillis();
     quicksort(a);
     stop = System.currentTimeMillis();
     elapsed = (stop - start) / 1000.0;
     System.out.println("Quicksort:   " + elapsed + " seconds");

     // print statistics
     System.out.println("Comparisons: " + comparisons);
     System.out.println("Exchanges:   " + exchanges);
 }
}
```

Source: http://www.cs.princeton.edu/introcs/42sort/QuickSort.java.html

Haskell

Lazy evaluation

- Allow infinite lists
- Expressions only evaluated if needed

Haskell

Some list capabilities:

```
main :: IO ()
main =  do
        print([1,3..])
```

Keeps going infinitely…

In practice lazy; can use as much as you want

Haskell

Lazy evaluation – let's run some code

```
squares = [n*n | n <- [0..]]
member n (m:x)
  | m<n = member n x
  | m==n = True
  | otherwise = False

main :: IO ()
main =  do
  print(member 16 squares)

  print(member 15 squares)
```

Checking if number
can be expressed as n*n
[0,1,4,9,16,25,36,49,64,81,100,121…

Support for functional in imperative languages

- Anonymous functions (Lambda expressions)

Remember Scheme:

```
((lambda (a b) (+ a b)) 4 5)
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Javascript: named function

```
function name (formal parameters) {
  body
}
```

Javascript: name omitted function

```
function (formal parameters) {
body
}
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

If more than one parameter, then enclosed in parentheses

If system cannot infer type of parameters, may be preceded by name type

Return value type always inferred

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

Example:

```
int [] numbers = {-3,0,4,5,1,-6}
int [] positives = Array.FindAll(numbers, n=>n>0);
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

Example:

```
int [] numbers = {-3,0,4,5,1,-6}
int [] positives = Array.FindAll(numbers, n=>n>0);
```

used as parameter to a methods

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

Example:

```
int [] numbers = {-3,0,4,5,1,-6}
int [] positives = Array.FindAll(numbers, n=>n>0);
```

C# method searches an array; retrieves all elements that match condition

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

Example:

```
int [] numbers = {-3,0,4,5,1,-6}
int [] positives = Array.FindAll(numbers, n=>n>0);
```

So passing block of code to a method

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
parameters => expression

Example:

```
int [] numbers = {-3,0,4,5,1,-6}
int [] positives = Array.FindAll(numbers, n=>n>0);

// now, positives is 4,5,1
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

C#
Can also do named version:

Example:

```
Func <int,int,int> evall = (a,b) => 3*(a + b/2);
int result = evall(6,22);
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Java 8, similar to c#:
parameters -> expression

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
y=lambda a,b : 2*a-b
print(y(2,3))
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
def thepower(n):
    return lambda x: x**n
f = thepower(2)
print(f(8))
f = thepower(3)
print(f(8))
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
def thepower(n):
    return lambda x: x**n
f = thepower(2)
print(f(8))
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
def thepower(n):
    return lambda x: x**n
f = thepower(2)
print(f(8))
```

Returns 64

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
def thepower(n):
    return lambda x: x**n
f = thepower(2)
print(f(8))
f = thepower(3)
print(f(8))
```

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
def thepower(n):
    return lambda x: x**n
f = thepower(2)
print(f(8))
f = thepower(3)
print(f(8))
```

Returns 512

Support for functional in imperative languages

- Anonymous functions (like Lambda expressions) are part of Python, Javascript, Ruby, Java, C#

Python:

```
f = lambda x: print(x)
f("hi")
```

Support for functional in imperative languages

- More on Python

Higher order filter and map often use
lambda expressions as first parameter:

```
fib = [0,1,1,2,3,5,8,13,21,34,55]
result = filter(lambda x: x % 2 == 0, fib)

print(result)

list(result)
```

Support for functional in imperative languages

- More on Python

Higher order filter and map often use
lambda expressions as first parameter:

```
fib = [0,1,1,2,3,5,8,13,21,34,55]
result = filter(lambda x: x % 2 == 0, fib)

print(result)

list(result)
```

Returns all fib values divisible by 2

Support for functional in imperative languages

- More on Python

Higher order filter and map often use lambda expressions as first parameter:

```
list(map(lambda x: x/2,[2,4,6,8]))
```

Support for functional in imperative languages

- More on Python

Partial function application (like currying of Haskell)

```
from operator import add;
from functools import partial;
```

Need to import functional version of addition
Operator named add from operator module…

Support for functional in imperative languages

- More on Python

Partial function application (like currying of Haskell)

```
from operator import add;
from functools import partial;
add5 = partial(add,5);
add5(15)
```

Support for functional in imperative languages

- More on Python

Head and tail…

```
theList = [1, 2, 3, 4, 5]
head, *tail = theList
print(head)
print(tail)
```

Support for functional in imperative languages

- More on Python

Head and tail…

```
theList = [1, 2, 3, 4, 5]
head = theList[0]
tail = theList[1:]
print(head)
print(tail)
```

Support for functional in imperative languages

- More on Python

Head and tail… also:

```
theList = [1, 2, 3, 4, 5]
head, *tail = theList
print(head)
print(tail)

Print[head, *tail]
```

Let’s try append in Python functional style

Haskell:

```
append ([],lis2) = lis2
append(h:t,lis2) = h:append(t,lis2)
```

ML:

```
fun append ([],lis2) = lis2
| append(h::t,lis2) =
h::append(t,lis2)
```

Scheme:

```
(define (append lis1 lis2)
(cond
 ((null? lis1) lis2)
   (else (cons (car lis1)
     (append (cdr lis1) lis2)))
))
```

Let's try append in Python functional style

Python:

```
def append(list1, list2):
    if list1==[]:
      return list2;
    else:
      h,*t = list1;
      return ([h,append(t,list2)]);

print(lis1+lis2)
lis1= [1,2,3]
lis2= [4,5,6]
print(append(lis1,lis2))
```

Let’s try append in Python functional style

Python:

```
def append(list1, list2):
    if list1==[]:
      return list2;
    else:
      h,*t = list1;
      return ([h,append(t,list2)]);

print(lis1+lis2)
lis1= [1,2,3]
lis2= [4,5,6]
print(append(lis1,lis2))
```


But we are missing cons…

Summary functional in imperative

- Interesting that renewed interest in functional languages
- Mainly, functional capabilities in imperative languages in recent years

Summary functional in imperative

- Interesting that renewed interest in functional languages
- Mainly, functional capabilities in imperative languages in recent years
- Also interest from perspective of side effects and parallel computing

Comparison functional vs imperative

- Functional versus imperative???

Comparison functional vs imperative

- Functional can have simple syntactic structure (e.g., list structure of Scheme used for both code and data)

Comparison functional vs imperative

- Functional can have simple syntactic structure (e.g., list structure of Scheme used for both code and data)
  Syntax of imperative more complex

Comparison functional vs imperative

- Functional can have simple syntactic structure (e.g., list structure of Scheme used for both code and data)
  Syntax of imperative more complex

- Semantics of functional simpler and no side effects

Comparison functional vs imperative

- Functional can have simple syntactic structure (e.g., list structure of Scheme used for both code and data)
  Syntax of imperative more complex

- Semantics of functional simpler and no side effects

- Functional programming can increase productivity (as in smaller programs). See Haskell quicksort!

Comparison functional vs imperative

- Functional can have simple syntactic structure (e.g., list structure of Scheme used for both code and data)
  Syntax of imperative more complex

- Semantics of functional simpler and no side effects

- Functional programming can increase productivity (as in smaller programs). See Haskell quicksort!

- Execution efficiency: functional slower than imperative

Comparison functional vs imperative

- Reliability???

Comparison functional vs imperative

- Reliability??? Functional has no side effects. Therefore concurrent more natural for functional; since no side effects can divide into functions that are executed concurrently

Comparison functional vs imperative

- Readability???

Comparison functional vs imperative

- Readability???

Compare C code:

```
int sumCubes (int n) {
   int sum = 0;
   for (int index=1; index<=n; index++)
      sum+=index*index*index;
   return sum;
}
```

Comparison functional vs imperative

- Readability???

Compare C code:

```
int sumCubes (int n) {
   int sum = 0;
   for (int index=1; index<=n; index++)
      sum+=index*index*index;
   return sum;
}
```

To Haskell:

```
sumCubes n = sum(map(^3)[1..n])
```

Comparison functional vs imperative

- Readability???

Compare C code:

```c
int sumCubes (int n) {
   int sum = 0;
   for (int index=1; index<=n; index++)
      sum+=index*index*index;
   return sum;
}
```

To Haskell:

```haskell
sumCubes n = sum(map(^3)[1..n])
```

1. Build list [1..n]
2. Create new list mapping the cube of each element
3. Sum new list

Comparison functional vs imperative

- But why have functional languages not attained even greater popularity?

Comparison functional vs imperative

- But why have functional languages not attained even greater popularity?

Perhaps inefficiency of earlier implementations

Comparison functional vs imperative

- But why have functional languages not attained even greater popularity?

Perhaps inefficiency of earlier implementations

Most programmers learn imperative first – so functional might seem strange

Comparison functional vs imperative

- But why have functional languages not attained even greater popularity?

Perhaps inefficiency of earlier implementations

Most programmers learn imperative first – so functional might seem strange

But as noted, some features of functional making their way into imperative…

Programming paradigms

- Imperative
- Functional
- ??

Programming paradigms

- Imperative
- Functional
- Logical
  What is that?

Programming paradigms

- Imperative
- Functional
- Logical
  What is that?

Logical programs: declarative rather than procedural
Only desired results (and collections of facts and rules) specified, rather than detailed procedure for producing results

Programming paradigms

- Imperative
- Functional
- Logical
  What is that?

Logical programs: declarative rather than procedural
Only desired results (and collections of facts and rules) specified, rather than detailed procedure for producing Results

Syntax and semantics very different from imperative

Towards logical languages: applications

- What languages?

Towards logical languages: applications

- What languages?

We’ll learn Prolog

Towards logical languages: applications

- ???

Towards logical languages: applications

- Relational Database Management Systems
  e.g., Structured Query Database (SQL) is non procedural (tables of information; relations between tables)

Towards logical languages: applications

- Relational Database Management Systems
  e.g., Structured Query Database (SQL) is non procedural (tables of information; relations between tables)

- Expert systems
  Designed to emulate user expertise; lots of facts and relations in databases. Use inference rules to infer new facts. Example: with Prolog

Towards logical languages: applications

- Relational Database Management Systems
  e.g., Structured Query Database (SQL) is non procedural (tables of information; relations between tables)

- Expert systems
  Designed to emulate user expertise; lots of facts and relations in databases. Use inference rules to infer new facts. Example: with Prolog

  Theorem proving

  Recent example: IBM Watson won jeopardy challenge

Towards logical languages: applications

- Relational Database Management Systems
  e.g., Structured Query Database (SQL) is non procedural (tables of information; relations between tables)

- Expert systems
  Designed to emulate user expertise; lots of facts and relations in databases. Use inference rules to infer new facts. Example: with Prolog

  Theorem proving

  Recent example: IBM Watson won jeopardy challenge

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

Natural Language Processing With Prolog in the IBM *Watson* System

Adam Lally
IBM Thomas J. Watson Research Center
Paul Fodor Stony Brook University
24 May 2011

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

On February 14-16, 2011, the IBM Watson question answering system won the Jeopardy! Man vs. Machine Challenge by defeating two former grand champions, Ken Jennings and Brad Rutter. To compete successfully at Jeopardy!, Watson had to answer complex natural language questions over an extremely broad domain of knowledge. Moreover, it had to compute an accurate confidence in its answers and to complete its processing in a very short amount of time.

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

The Question-Answering (QA) problem requires a machine to go beyond just matching keywords in documents, which is what a web-search engine does, and correctly interpret the question to figure out what is being asked. The QA system also needs to find the precise answer without requiring the aid of a human to read through the returned documents.

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

To address these challenges, the research team at IBM developed a software architecture called DeepQA, on which Watson is implemented. The DeepQA architecture assumes and pursues multiple interpretations of the question, generates many plausible answers or hypotheses, collects evidence for these hypotheses, and evaluates the evidence to determine if it supports or refutes those hypotheses [2]. Watson contains hundreds of different algorithms that evaluate evidence along different dimensions.

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

Watson's NLP begins by applying a parser [5] that converts each text sentence into a more structured form: a tree that shows both surface structure and deep, logical structure. For example, in the following example Jeopardy! question:

POETS & POETRY: He was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

Watson's NLP begins by applying a parser [5] that converts each text sentence into a more structured form: a tree that shows both surface structure and deep, logical structure. For example, in the following example Jeopardy! question:

POETS & POETRY: He was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907

The output of the parser includes, among many other things, that "published" is a verb with base form (or *lemma)* "publish", subject "he", and object "Songs of a Sourdough".

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

Next, Watson applies numerous detection rules that match patterns in the parse. These rules detect elements such as the *focus* of the question (the words that refer to the answer, in this case “he”), the *lexical answer types* (terms in the question or category that indicate what type of entity is being asked for, in this case “poet”), and the relationships between entities in either a question or a potential supporting passage.

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

We required a language in which we could conveniently express pattern matching rules over the parse trees and other annotations (such as named entity recognition results), and a technology that could execute these rules very efficiently. We found that Prolog was the ideal choice for the language due to its simplicity and expressiveness. The information in the parse is easily converted into Prolog facts, such as (the numbers representing unique identifiers for parse nodes):

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

The information in the parse is easily converted into Prolog facts, such as (the numbers representing unique identifiers for parse nodes):

```
lemma(1, "he").
partOfSpeech(1,pronoun).
lemma(2, "publish").
partOfSpeech(2,verb).
lemma(3, "Songs of a Sourdough").
partOfSpeech(3,noun).
```

Towards logical languages: applications

Fairly recent example: IBM Watson won jeopardy challenge

https://www.cs.miami.edu/home/odelia/teaching/csc419_spring20/syllabus/IBM_Watson_Prolog.pdf

Such facts were consulted into a Prolog system and several rule sets were executed to detect the focus of the question, the lexical answer type and several relations between the elements of the parse. A simplified rule for detecting the `authorOf` relation can be written in Prolog as follows:

```
authorOf(Author,Composition) :-
    createVerb(Verb),
    subject(Verb,Author),
    author(Author),
    object(Verb,Composition),
    composition(Composition).

createVerb(Verb) :-
    partOfSpeech(Verb,verb),
    lemma(Verb,VerbLemma),
    member(VerbLemma, ["write", "publish",...]).
```