15-150 Fall 2015
Homework 02

Out: Wednesday, 9 September 2015
Due: Tuesday, 15 September 2015 at 23:59 EST

1 Introduction

In this assignment, you will go over some of the basic concepts we want you to learn in this course, including defining recursive functions and proving their correctness. We expect you to follow the methodology for defining a function, as shown in class.

1.1 Getting The Homework Assignment

The starter files for the homework assignment have been distributed through our `git` repository, as usual.

1.2 Submitting The Homework Assignment

Submissions will be handled through Autolab, at

```
https://autolab.cs.cmu.edu
```

In preparation for submission, your `hw/02` directory should contain a file named exactly `hw02.pdf` containing your written solutions to the homework.

To submit your solutions, run `make` from the `hw/02` directory (that contains a `code` folder and a file `hw02.pdf`). This should produce a file `hw02.tar`, containing the files that should be handed in for this homework assignment. Open the Autolab web site, find the page for this assignment, and submit your `hw02.tar` file via the "Handin your work" link.

The Autolab handin script does some basic checks on your submission: making sure that the file names are correct; making sure that no files are missing; making sure that your code compiles cleanly. Note that the handin script is *not* a grading script—a timely submission that passes the handin script will be graded, but will not necessarily receive full credit. You can view the results of the handin script by clicking the number (usually either 0.0 or 1.0) corresponding to the "check" section of your latest handin on the "Handin History" page. If this number is 0.0, your submission failed the check script; if it is 1.0, it passed.

Remember that your written solutions must be submitted in PDF format—we do not accept MS Word files or other formats.

Your `hw02.sml` file must contain all the code that you want to have graded for this assignment, and must compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

1.3 Due Date

This assignment is due on Tuesday, 15 September 2015 at 23:59 EST. Remember that you may use a maximum of one late day per assignment, and that you are allowed a total of three late days for the semester.

1.4 Methodology

You must use the five step methodology discussed in class for writing functions, for **every** function you write in this assignment. Recall the five step methodology:

1. In the first line of comments, write the name and type of the function.
2. In the second line of comments, specify via a `REQUIRES` clause any assumptions about the arguments passed to the function.
3. In the third line of comments, specify via an `ENSURES` clause what the function computes (what it returns).
4. Implement the function.
5. Provide testcases, generally in the format
   `val <return value> = <function> <argument value>`.

For example, for the factorial function:

```
(*  fact : int -> int
 *  REQUIRES:  n >= 0
 *  ENSURES: fact(n) ==> n!
*)

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

(* Tests: *)

val 1 = fact 0
val 720 = fact 6
```

2 Basics

ML has a built-in function `trunc : real -> int` that computes the integer part of a real number; for example, `trunc 3.9 = 3`. Consider the following ML code template:

```
fun pow (k : int, x : int) : int =
   if k = 0 then 1 else x * pow (k-1, x);

fun cube (y : real) : real = y * y * y;

fun bop1 (z : real) = pow (3, trunc (cube z));
fun bop2 (z : real) = cube (cube z);
fun bop3 (z : real) = pow (3, cube (trunc z));
```

Task 2.1 (6 pts). What are the types of the functions defined here? If not well typed, say so and explain briefly why.

Task 2.2 (4 pts). Give a step-by-step evaluation sequence for the expression

```
pow(1, 21 + 21)
```

that shows the order in which sub-expressions get evaluated and shows the syntactic value `v` to which the expression evaluates. A conditional expression `if e then e1 else e2` evaluates the test expression `e` first to a truth value, then selects the appropriate branch; a pair expression evaluates from left to right.

3 Recursive Functions

3.1 Fun Fact

Consider the function `fact` of type `int -> int`, given by:

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

The form of this recursive function definition gives us the following equivalences:

(1) `fact(0)` $=_{int}$ `1`

(2) `fact(n)` $=_{int}$ `n * fact(n-1),` for any integer `n` with `n>0`.

In particular, the following two instances of (2) are obtained by picking `n=1` and `n=2`:

- `fact(1)` $=_{int}$ `1 * fact(1-1)`
- `fact(2)` $=_{int}$ `2 * fact(2-1)`

Recall that *referential transparency* implies that the value of any expression is unchanged if we replace a sub-expression by another expression with the same value. Hence, by referential transparency and the facts that `1-1 = 0` and `2-1 = 1`, we can deduce the equivalences

(a) `fact(1)` $=_{int}$ `1 * fact(0)`

(b) `fact(2)` $=_{int}$ `2 * fact(1)`

Task 3.1 (5 pts). Using extensional equivalence and referential transparency, show that

$$\texttt{fact(3)} =_{int} \texttt{6}$$

Your answer must *not* use the $\Longrightarrow$ notation (for evaluation). You should instead properly use the equivalences (1) and (2) as well as (a) and (b) mentioned above. Cite these equivalences to justify your reasoning. Write your proof mathematically, line by line, justifying each step. Do not write an English paragraph.

Task 3.2 (4 pts). Consider the function `fact` above. Is it true that

$$\texttt{fact}(\sim \texttt{1}) =_{\texttt{int}} \texttt{fact}(\sim \texttt{2})?$$

Explain.

3.2 Primality Test

You have probably heard about prime numbers in the past. Here, we will be utilizing recursion to determine if a natural number is prime. Recall the definition for prime numbers:

Theorem 1. *A natural number $n > 1$ is prime if and only if it is divisible by only itself and 1.*

Given the input number n, a simple test for primality is to go through all of the numbers from 2 to $n - 1$ and check if each divides into n. This can be done with recursion, using an extra argument that keeps track of the current divisor that we should check next. We can test for divisibility using `mod`, because n is divisible by m if and only if n `mod` $m = 0$.

Of course, we only really need to check for divisibility by 2 through $\sqrt{n}$, but we will not penalize you if your code checks for divisibility by 2 through $n - 1$.

Task 3.3 (16 pts). Write an ML function

```
is_prime : int -> bool
```

in `hw02.sml` such that for all natural numbers `n`, `is_prime` returns true if `n` is prime and false otherwise.

Hint: Use a recursive helper function of a suitable type, as outlined above. Make sure you give proper documentation for any helper function(s) that you define, including type and specification.

4 Gregorian functional programming

Consider the following function definitions, based on an algorithm of Gauss for computing the day-of-the-week for a given date in the Gregorian calendar. (You don't need to understand why or if Gauss's algorithm is correct!)

As usual we represent a date as a triple consisting of a day number, a month number, and a year number.

```
fun gauss1 (D:int, M:int, Y:int) : int =
let
   val m = if M>2 then M-2 else M+10
   val y = if M>2 then Y else Y-1
   val c = y div 100
   val a = y mod 100
   val b = (13 * m - 1) div 5 + (a div 4) + (c div 4)
in
   (a + b + D - 2 * c) mod 7
end;

fun gauss2 (D:int, M:int, Y:int) : int =
let
   val m = (M + 9) mod 12 + 1
   val y = if M>2 then Y else Y-1
   val c = y div 100
   val a = y mod 100
   val b = (13 * m - 1) div 5 + (a div 4) + (c div 4)
in
   (a + b + D - 2 * c) mod 7
end;
```

Task 4.1 (2 pts). For each of the following expressions, what is its type? If the expression is not well typed, say why.

- `gauss1`
- `gauss1 (2,2,2015)`
- `gauss1 (1+1, 3-1, 5*403)`
- `gauss2 (1.0, 2.0, 2015.0)`

Task 4.2 (4 pts). For each of the above expressions, what is its value according to the ML runtime system? If the expression does not have a value, say so.

Task 4.3 (4 pts). For each of the above expressions, what syntactic value does it evaluate to, according to the evaluation rules for `=>*` given in class? Again, if the expression doesn't evaluate, say so. No need to give detailed justification.

Task 4.4 (12 pts). Write a function

```
is_valid_date : int * int * int -> bool
```

such that for all integer values `d, m, y`, `is_valid_date (d, m, y)` evaluates to `true` if

$$1 \leq \texttt{d} \leq n \;\&\; 1 \leq m \leq 12 \;\&\; y \geq 1582,$$

where n is the number of days in the $\texttt{m}^{th}$ month of year `y` of the Gregorian calendar; and `is_valid_date (d, m, y)` evaluates to `false` otherwise. (We only care about when $y \geq 1582$ because that was the year in which the Gregorian calendar was adopted.)

HINT: You will need to introduce helper functions for finding things like the length of month `m` of year `y` (the number of days in the month), and whether or not year `y` was a Leap year. The following quotes from Wikipedia are relevant:

> Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the year 2000 is.
>
> Thirty days hath September, April, June, and November. All the rest have thirty-one, Excepting February alone, Which hath but twenty-eight days clear, And twenty-nine in each leap year.

As usual we represent January as the first month. For example, `(1,2,2015)` means February 1, 2015.

Task 4.5 (6 pts). Assuming you have `is_valid_date` as in the previous question, which of the following statements is/are true? Give brief reasons.

- `gauss1` and `gauss2` are extensionally equivalent.
- `gauss1 (d, m, y) = gauss2 (d, m, y)` for all integer values `d, m, y`.
- `gauss1 (d, m, y) = gauss2 (d, m, y)` for all integer values `d, m, y` such that `is_valid_date (d, m, y) = true`.

Task 4.6 (6 pts). Write a function

```
start_of_next_month : int * int * int -> int * int * int
```

such that for all integer values `d, m, y` such that

```
is_valid_date (d, m, y) = true,
```

we have

```
start_of_next_month (d, m, y) = (1, m', y'),
```

where month `m'` of year `y'` is the month immediately after month `m` of year `y` in the Gregorian calendar.

5 Sums of powers

Task 5.1 (21 pts). When $n \geq 1$, the sum of the first n fourth powers, written in math notation as $\sum_{k=1}^{n} k^4$, or more informally as $1^4 + 2^4 + \cdots + n^4$, is equal to the value of

$$(6n^5 + 15n^4 + 10n^3 - n)/30.$$

Prove this, by induction on n. You can use without proof well-known facts such as

$$\begin{aligned}(n+1)^2 &= n^2 + 2n + 1\\ (n+1)^3 &= n^3 + 3n^2 + 3n + 1\\ (n+1)^4 &= n^4 + 4n^3 + 6n^2 + 4n + 1\\ (n+1)^5 &= n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1\end{aligned}$$

Consider the following ML functions:

```
fun pow (k:int, x:int) : int =
   if k=0 then 1 else x * pow(k-1, x)

fun check n =
   (6 * pow(5, n) + 15 * pow(4, n) + 10 * pow(3, n) - n) div 30
```

Task 5.2 (2 pts). How many multiplication operations get done when evaluating `pow (k, n)`?

Task 5.3 (2 pts). How many multiplications get done when evaluating `check n`?

Task 5.4 (6 pts). Rewrite the `check` function to reduce the number of multiplication operations by a factor of 2 or more.