Programming Exercises 1: Big Integers

(Submissions only via Moodle)

August 19, 2014

Consider the following data types:

```
type nat = O | S of nat;;


type sign = Pos | Neg;;

type digitseq = int list;;

type bigint = sign*int*digitseq;;
```

A "big integer" can be represented as a triple consisting of the sign (`Pos` or `Neg`), an integer greater than 0 representing the base r, and a sequence of integers that represent digits (which should be in the range $0 \leq d_i < r$, written in least-significant-digit to most-significant-digit order. The most-significant-digit should not be a 0.

1. Write a program `maxint: unit -> int`, that reports the largest `int` value that OCaml permits. [Hint: keep doubling the value until there is an overflow].

2. Write a program `check_bigint: bigint -> bool`, that given a triple in the `bigint` format, checks that it is a valid representation of an integer as outlined above.

3. Write a program `int2bigint: int * int -> bigint`, that converts a given OCaml int n and a base r into a `bigint` in base r. You need to check that r is a valid base, raising an exception `InvalidBase` otherwise

4. Write a program `bigint2int: bigint -> int`, that converts a given `bigint` into an OCaml `int`.

5. *Composing* the programs that convert between `nat` and `int`, write programs `nat2bigint: nat * int -> bigint` and `bigint2nat: bigint -> nat`, that convert between the `bigint` and `nat` types, raising exceptions when appropriate.

6. Write a program `bigplus: bigint * bigint -> bigint`, that adds two given "big integers".

7. Write a program `bigtimes: bigint * bigint -> bigint`, that multiplies two given "big integers".

You must document your programs adequately, and provide enough test inputs on which your programs run.