Assignment 4: Implementation of BigInt Package

March 23, 2016

1 Introduction

Language PL0 provides two basic data types, int and bool, which represents integer and boolean respectively. Boolean can take two values as tt and ff, where as integers does not have any limit or any constraints such as number of bytes used, to represent integer. But, language using which we are constructing compiler (such as SML, Ocaml, Haskell), may have some limitations, so we have to overcome those limitations for integer.

So, basically you have to create package/library, BigInt, which overcomes such limitations and can do operations on any arbitrarily large integers. For example, it should be able to perform,

1245156215336613646 * 1656100031608652335215

without overflow and should produce proper output.

2 BigInt Package Specifications

You have to implement following functions for BigInt package:

signature BigInt =

sig

```
type bigint
val getbigint: int → bigint
val bi2str : bigint → string
val str2bi : string → bigint
val lt : bigint * bigint → bool
val leq : bigint * bigint → bool
val gt : bigint * bigint → bool
val geq : bigint * bigint → bool
val eq : bigint * bigint → bool
val neq : bigint * bigint → bool
```

```
    val div : bigint * bigint → bigint
    val mul : bigint * bigint → bigint
    val add : bigint * bigint → bigint
    val sub : bigint * bigint → bigint
    val mod : bigint * bigint → bigint
    val unminus : bigint → bigint
end;
```

2.1 Description of Functions

Note that operators which are to be implemented on 'bigint' are same operators, which are defined in language PL0.

Explanation of above functions are as follows:

- val getbigint: int → bigint

  Takes one argument of type 'int', which is defined by language on which you are implementing (such as SML, Ocaml and Haskell) and converts it to type 'bigint'

- val bi2str : bigint → string

  Takes one argument of type 'bigint' and returns its corresponding string representation. This function will be helpful in printing results.

- val str2bi : string → bigint

  Takes one argument of type 'string' and converts it to corresponding 'bigint' representation. This function will be helpful in taking inputs.

- val lt : bigint * bigint → bool

  This function implements relational operator '<' (less than operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a < b', which can be either true or false.

- val leq : bigint * bigint → bool

  This function implements relational operator '≤' (less than or equal to operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a ≤ b', which can be either true or false.

- val gt : bigint * bigint → bool

  This function implements relational operator '>' (greater than operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a > b', which can be either true or false.

- val geq : bigint * bigint → bool

  This function implements relational operator '≥' (greater than or equal to operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a≥b', which can be either true or false.

- val eq : bigint * bigint → bool

  This function implements relational operator '=' (equal to operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a = b', which can be either true or false.

- val neq : bigint * bigint → bool

  This function implements relational operator '<>' (not equal to operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a <> b', which can be either true or false.

- val div : bigint * bigint → bigint

  This function implements arithmetic operator '/' (division operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a / b', which is of type 'bigint'.

- val mul : bigint * bigint → bigint

  This function implements arithmetic operator '*' (multiplication operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a * b', which is of type 'bigint'.

- val add : bigint * bigint → bigint

  This function implements arithmetic operator '+' (addition operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a + b', which is of type 'bigint'.

- val sub : bigint * bigint → bigint

  This function implements arithmetic operator '-' (subtraction operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a - b', which is of type 'bigint'.

- val mod : bigint * bigint → bigint

  This function implements arithmetic operator '%' (modulo operator). Takes two arguments, lets say a and b respectively, of type 'bigint' and returns result of 'a % b', which is of type 'bigint'.

- val unminus : bigint → bigint

  This function implements Unary minus. Takes one argument of type 'bigint', lets sat a, and returns result of '$\sim a$', which is of type 'bigint'

2.2 Note:

- This BigInt package will not give any errors like overflow, etc. Only division by zero is major error to be reported.

- You can code either in SML or OCaml or Haskell, but imperative paradigm such as "for" loops, "while" loops, etc. are strictly prohibited.

- Your assignment will be autograded. So we will be making call to these functions to evaluate, so make sure that you do NOT change function names and keep it as it is given to you.

3 Instructions for Submission

- **All submissions must be through moodle. No other form of submission will be entertained.**
- **No submissions will be entertained after the submission portal closes.**
- **Sometimes there are two deadlines possible – the early submission deadline (which we may call the "lifeline") and the final "deadline". All submissions between the "lifeline" and the "deadline" will suffer a penalty to be determined appropriately.**

4 What to Submit?

- You will create one folder which will have 2 files, program file and the writeup file.
- The program file should be named with your Kerberos ID. For example, if kerberos id is 'cs1140999' then the file name should be cs1140999.sml or cs1140999.ocaml. The writeup should be named as "writeup.txt".
- Both the files should be present in one folder. Your folder also should be named as your Kerberos ID. For example, if kerberos id is 'cs1140999' then the folder should be called cs1140999.
- The first line of writeup should contain a numeral indicating language preferred with 0-ocaml, 1-sml and 2-haskell.
- For submission, the folder containing the files should be zipped(".zip" format). Note that, you have to zip folder and NOT the files.
- This zip file also should have name as your Kerberos ID. For example, if kerberos id is 'cs1140999' then the zip file should be called cs1140999.zip.
- Since the folder has to be zipped the file cs1140999.zip should actually produce a new folder cs1140999 with files (cs1140999.sml or cs1140999.ocaml) and writeup.txt.

  Hence the command *"unzip -l cs1140999.zip"* should show

  cs1140999/cs1140999.sml
  cs1140999/writeup.txt

- After creating zip, you have to convert ".zip" to base64(.b64) format as follows (for example in ubuntu):
  base64 cs1140999.zip > cs1140999.zip.b64 will convert .zip to .zip.b64
  This cs1140999.zip.b64 needs to be uploaded on moodle.
- After uploading, please check your submission is up-to the mark or not, by clicking on evaluate. It will show result of evaluation. If folder is as required, there will be no error, else REJECTED with reason will be shown. So, make sure that submission is not rejected.