CS 270 Lab 3 (Bits, bits and more bits)

Week 3 - Oct. 9 – Oct. 13, 2017.

Name 1:____

Drexel Username 1:____

Name 2:____

Drexel Username 2:____

Name 3:____

Drexel Username 3:____

Grading:

Part 1 (50%) ____

Part 2 (50%) ____

Instructions: For this exercise you are encouraged to work in groups of two or three so that you can discuss the problems, help each other when you get stuck and check your partner's work. There are two parts.

In this lab, students implement recursive functions to add and multiply two numbers represented in binary.

Binary numbers are represented by a list of bits [zeros or ones].

(b_0 b_1 ... b_{n-1}) represents b = b_0 + b_1*2 + ... + b_{n-1}*2^{n-1}

For example, the binary representation for 13 is 1101 and is represented by the list (1 0 1 1). The number zero is represented by the empty list. Note that representations are not unique. E.G. 0 = () and also (0 0 0). We will assume binary numbers are normalized. I.E. there are no leading zeros. Normalized binary numbers are unique.

In part 1 students implement binary addition using the following recursive construction.

a = (a_0 a_1 ...a_{m-1}) and b = (b_0 b_1 ... b_{n-1})

a + b = a_0 + b_0 + 2*[a' + b'], where a' = (a_1 ...a_{m-1}) and b' =(b_1 ... b_{n-1}).

The multiplication by 2 after the recursive call corresponds to a shift and consequently the result of the a'+b' is one position to the right of the sum of a_0 and b_0. Thus the sum of a_0 and b_0 may simply be cons'd onto the result of the recursive call.

When adding the bits a_0 and b_0 there may be a carry. I.E. when both a_0 and b_0 are 1 the result is 2. Thus the low order bit of the result is 0 and 1, the carry, must be added to the recursive sum $a'+b'$. In general when adding two bits and a carry the maximum result is 3 which implies the sum bit is either 0 or 1 and the carry is also either 0 or 1.

Thus when implementing binary addition we include an extra input equal to the carry in, called cin, and compute (binadd cin a b) = cin + a + b. The above recursive construction becomes

$a + b + cin = (a_0 + b_0 + cin) + 2*[a'+b']$ which is equal to

$c_0 + 2*[cout + a' + b']$, where $c_0 = (a_0 + b_0 + cin) \bmod 3$ and

$cout = 1$ when $(a_0 + b_0 + cin) > 1$ and 0 otherwise.

In part 2 students implement binary multiplication using the following recursive construction.

$a*b = a_0*b + 2*(a_1 \ldots a_{m-1})*b$

Multiplication by a power of two is easy when numbers are in binary. Multiplication by 2^h shifts the number h places to the right which corresponds to prepending h leading zeros.

$2^h*b = (0 \ldots 0\ b_0\ b_1 \ldots b_{n-1})$

In order to implement the above recursive multiplication algorithm, you will need to implement a helper function binmult2 which multiplies a binary number by a power of two.

As an aid to implementing these functions you may wish to use the let* special form to introduce names for temporary values that are used later. The syntax (let* (b1 ... bt) exp) introduces bindings b1,...,bt which can be used in exp which is the result that is returned. let* as compared to let allows preceding bindings to be used in the computation of the values for the subsequent bindings.

E.G. (let* ((x 1) (y (+ x 3))) (+ x y)) returns 5