York University
CSE 4111 Fall 2009
Instructor: Jeff Edmonds

Family Name: ____________________ Given Name: ____________________

Student #: ____________________ Email: ____________________

The following are problems that Jeff made up likely won't every use often because the may be too hard. Maybe the better students might enjoy them too.

1. Adding:

   (a) Define the function $\sharp : \{0,1\}^3 \Rightarrow \{0,1\}^2$ so that $\sharp(x,y,z)$ gives in binary the number of bits of x, y, and Z that are one. For example, $\sharp(0,0,0) = 00$, $\sharp(0,0,1) = 01$, $\sharp(0,1,0) = 01$, $\sharp(1,0,1) = 10$, $\sharp(1,1,0) = 10$, and $\sharp(1,1,1) = 11$. Design a circuit with three inputs and two outputs using AND, OR, and NOT gates that computes $\sharp$. Let $\sharp_1(x,y,z)$ be the high order bit and $\sharp_2(x,y,z)$ be the low order bit. It might be easier to separately build circuit for $\sharp_1$ and for $\sharp_2$. More marks will be given if fewer gates are used.

   (b) Recall how to add to binary numbers. 1011 + 1101 = 11000. Add the following binary numbers. 100101100100 + 011001011100.

   (c) Do Sipser question 9.15, building a circuit that adds two binary numbers. In addition to AND, OR, and NOT circuits, you can build the circuit using $\sharp$ gates. Denote the one input with $x_{n-1}, \ldots, x_0$, the other with $y_{n-1}, \ldots, y_0$, and the sum with $z_n, \ldots, z_0$.

   (d) Design a three tape TM that adds two binary numbers. Initially, the first and second tapes contain the two numbers. In end, the third tape should contain the sum. You can assume that the two inputs and the output all have the same length, i.e. the high order bits are zero. Describe it using the implementation level description (pg 145). However, I am particularly interested in how the transition function $\delta(q_i, x, y, z) = (q_j, x', y', z', R, R, R)$ is defined as the bits are being added together. You may want to use $\sharp_1$ and $\sharp_2$, when defining the transition function.

2. Faster primitive recursive programs

   (a) It is a royal pain that primitive recursive programs on input $\langle x, y, z\rangle$ can only recurse by subtracting one on one argument, i.e. on $\langle x, y-1, z\rangle$ but won't allow you to recurse by dividing by two, i.e. on $\langle x, \lfloor \frac{y}{2} \rfloor, z\rangle$ or on more than one argument, i.e. on $\langle x, \lfloor \frac{y}{2} \rfloor, z-1\rangle$. It would be really nice to be able to say that $F(x,y)$ is "primitive recursive" if H and G are and

   $F(x,y,z) = H(x,y,z,F(x,\lfloor \frac{y}{2} \rfloor), z-1)$
   $F(x,0,0) = G(x)$.

   For example $Log(y) = Log(\lfloor \frac{y}{2} \rfloor) + 1$.

   This problem is to design a generic way for computing this as a primitive recursive function.

Towards this goal, define $R = \max(\lfloor \log y \rfloor + 1, z)$ to be the number of times that the algorithm needs to recurses from input $\langle x, y, z\rangle$ until it reaches the base case $\langle x, 0, 0\rangle$. (Remember neither $\frac{y}{2}$ nor $z-1$ ever goes negative.)

Define $\langle x, Smaller_y(i), Smaller_z(i)\rangle$ to be the input that the stackframe i iterations from the bottom is given and define $f(x, y, z, i) = F(x, Smaller_y(i), Smaller_z(i))$ to be the answer that this stack frame returns.

For example, for $i = 0$, the input is the base case $\langle x, Smaller_y(0), Smaller_z(0)\rangle = \langle x, 0, 0\rangle$. When $i = 1$, this input might be $\langle x, 1, 0\rangle$ or $\langle x, 0, 1\rangle$ depending on which parameter reached zero first. Finally when $i = R$, $\langle x, Smaller_y(R), Smaller_z(R)\rangle$ is the original instance $\langle x, y, z\rangle$. It follows that $f(x, y, z, R) = F(x, y, z)$, which is what we are trying to compute.

$Smaller_y(i) = High(y, i) = \lfloor \frac{y}{2^{R-i}} \rfloor$ is the integer consisting of the i highest order bits of y. For example $11 = 1011_2$, so $High(11, 2) = \lfloor \frac{11}{2^{\lfloor 4 \rfloor - 2}} \rfloor = \lfloor \frac{11}{4} \rfloor = 2 = 10_2$. Note that as needed $\lfloor \frac{High(y,i)}{2} \rfloor = High(y, i-1)$.

It is more than you need to know, but $Smaller_z(i) = \min(i, \max(z - R + i, 0))$. If z is the last parameter to reach zero, then $R = z$ and $Smaller_z(i) = i$, giving as required $Smaller_z(0) = 0$ and $Smaller_z(R) = z$. On the other hand, if z reaches zero earlier, then $R > z$ and $Smaller_z(i) = \max(z - R + i, 0)$, giving as required $Smaller_z(0) = Smaller_z(1) = 0$ and $Smaller_z(R) = z$.

Recall that our goal is to prove that F is primitive recursive. It is sufficient to prove instead that f is primitive recursive because if it is, then so is F because $F(x, y) = f(x, y, R)$.

Note that $Smaller_y(i)$ and $Smaller_z(i))$ are primitive recursive using things learned in class. Assume that H and G are primitive recursive. Use them to write a primitive recursive function for f. The answer is two quick lines. All you need to use are

$F(x, y, z) = H(x, y, z, F(x, \lfloor \frac{y}{2} \rfloor, z-1))$

$F(x, 0, 0) = G(x)$

$f(x, y, z, i) = F(x, Smaller_y(i), Smaller_z(i))$.

(b) Lets use this faster recursive structure to compute $Add(x, y)$. The form of the recursion will be

$Add(x, y) = H_{Add}(x, y, Add(\lfloor \frac{x}{2} \rfloor, \lfloor \frac{y}{2} \rfloor))$

$Add(0, 0) = 0$

Your goal is to define H_{Add} defining $Add(x, y)$ using the known primitive recursive functions $Double(z) = 2z$ and $Odd(x)$ which is 1 if x is odd and 0 if it is even.

Hint start with $x = 2a$ or $x = 2a + 1$ and $y = 2b$ or $y = 2b + 1$.

You must also prove by induction on the size of the input that $Add(x, y)$ is defined correctly.

(c) We will now use this faster recursive structure to compute $Mult(x, y)$. The form of the recursion will be

$Mult(x, y) = H_{Mult}(x, y, Mult(x, \lfloor \frac{y}{2} \rfloor))$

$Mult(0, 0) = 0$

In addition to $Double(z) = 2z$ and $Odd(x)$, you have $Add(x, y)$ available to you. Again prove by induction that $Mult(x, y)$ is defined correctly.

(d) Now lets bound the time to compute $Add(x, y)$ and $Mult(x, y)$.

$Smaller_y(i) = High(y, i) = \lfloor \frac{y}{2^{R-i}} \rfloor$ just involves shifting bits in the binary expansion of y, so it is reasonable to say that this can be done in constant time. Same for $Double(x)$ and $Odd(x)$. Give recurrence relations on the time $T_{Add}(n)$ needed to add two n bit numbers and the time $T_{Mult}(n, m)$ needed to multiply x and y when x has n bits and y has m.