CpSc 513 **Homework 2** Due: Dec. 6, 2016, 11:59pm

Instructions

Please submit your solution by e-mail to mrg@cs.ubc.ca.

- Please submit a file called `hw2.txt` or `hw2.pdf` that has your solutions to the problems that don't involve writing python code. Please submit `add.py`, stamps.py, and not3.py for your source code from problems 2 through 4. You can combine all of these into a `.zip` or `.tgz` file. Thanks.

The Questions

1. Finishing the interpolants from resolution proof proof. (45 points)
   Let A and B be two propositional formulas represented as sets of clauses. Thus, we can write $A \wedge B$ to indicate their conjunction, or, equivalently, $A \cup B$ to indicate the union of the clause sets. Let $Interp$ be the proposed interpolant computed using the method described in "Interpolation and SAT-Based Model Checking". In class, we proved $A \Rightarrow B$. In this problem, you will prove that $Interp \wedge B$ is unsatisfiable. You can do this anyway you like. However, the remainder of this problem statement sketches a proof that your are welcome (and even encouraged) to follow.

   The construction of $Interp$ was based on a the graph representation of a resolution proof that $A \cup B$ is unsatisfiable. The graph is a DAG with one root vertex for each clause of $A \cup B$, and one leaf vertex. Each vertex is labled with a clause, c. The root vertices are labled with their clauses from A or B. Each non-root vertex, v, has two parents, v_1, and v_2. The clause for v is the clause obtained by resolution of the clauses for v_1 and v_2. From now on, we will designate each vertex by its associated clause.

   To construct $Interp$, each vertex, c, received an additional label $p(c)$ as described in the paper. For my suggested proof, we can add one more label, $q(c)$. If $c \in A$, then $q(c) = T$. If $c \in B$ then $q(c) = c$. If c is a non-root vertex with parents c_1 and c_2, then $q(c) = q(c_1) \wedge q(c_2)$.

   (a) (20 points) Show, by induction on the structure of the graph, that for each vertex, c, $B \rightarrow q(c)$.

   (b) (20 points) Show, by induction on the structure of the graph, that for each vertex, c, $(p(c) \wedge q(c)) \Rightarrow c$.

   (c) (5 points) Let c_{end} be the final (i.e. leaf) vertex of the DAG. By the assumption that $A \wedge B$ is unsatisfiable, c is the empty clause, i.e., it represents the proposition false. The rest is simple. Show that $p(c_{end}) \wedge q(c_{end})$ must be unsatisfiable. Noting that $Interp = p(c_{end})$, show that $Interp \wedge B$ must be unsatisfiable.

2. Equivalence checking (**25 points**) Figure 1 shows four implementations of a one-bit full adder. Two of them are correct, and two are not. We can SAT solving to verify the correct adders and find counter-examples for the others. For this problem, you will use Z3 as the SAT solver. To do so, I'm providing a module add.py. that includes the function `addCheck` which prints 'proved' if the adder is verified and otherwise prints a counter-example. I give an example with the function `test1` that tests the first adder. It passes. You can try changing it, for example remove a `Not`, change an `And` to an `Or`, or similar, and show that you get a counter-example.

   (a) (**15** points) I just showed that circuit 1 is a correct, one-bit, full adder. Write python descriptions of circuits 2, 3, and 4, and add them to add.py. Identify the correct adders and the buggy ones. For the buggy ones, report the counter-examples.

   (b) (**10** points) Should you believe my `addCheck` function? In Z3, you can convert boolean expressions to integers using an if-then-else expression. The form is `If(cond, then_expr, else_expr)`. Use this to verify that if `a`, `b`, `c_in`, `s`, and `c_out` satisfy `addCheck`, then `a + b + c_in == s + 2*c_out`. Show your code.

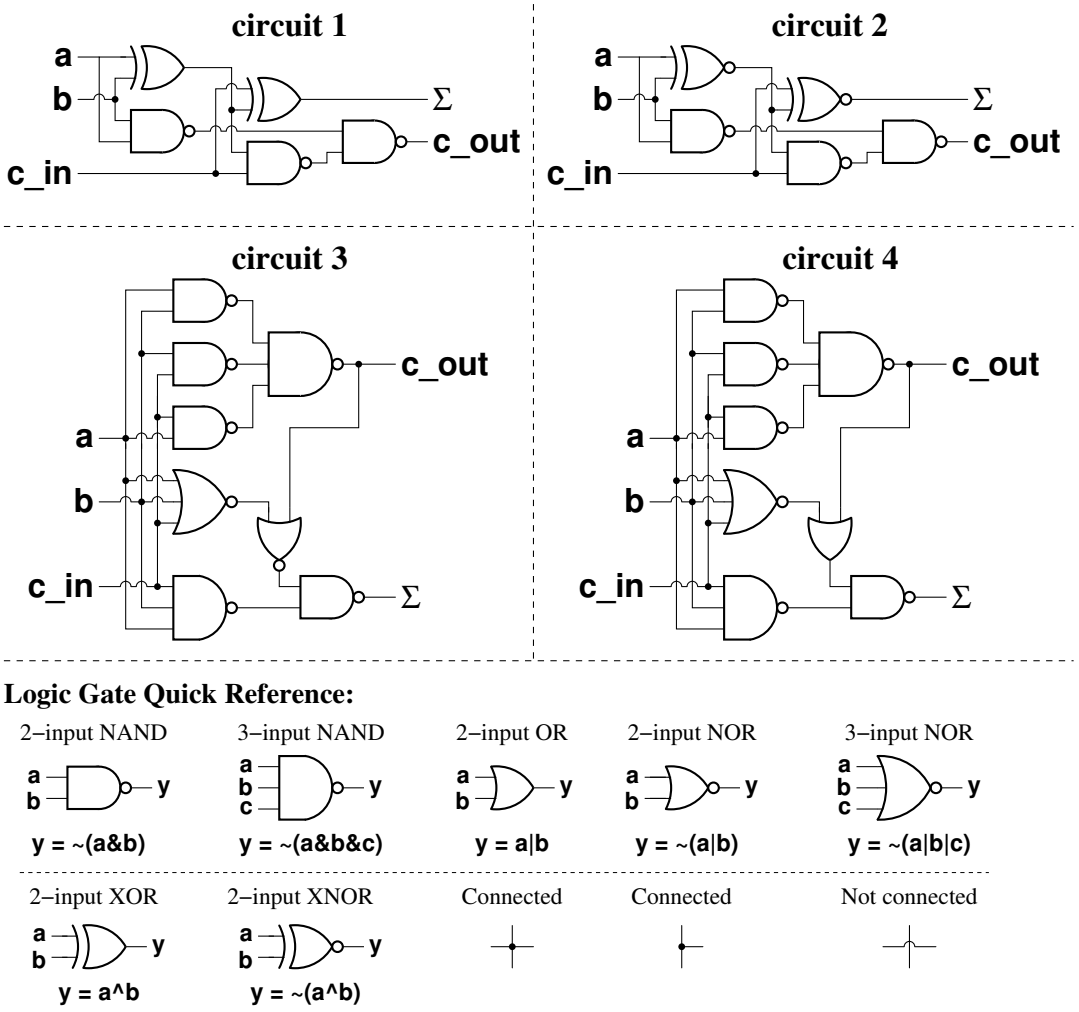

Figure 1: Four adder circuits (but only two work)

3. Postage (**25 points**)

Consider a problem of sending a letter or package that requires postage of p cents, and where the only stamps for sale come in amounts of s_0, s_0, $\ldots s_{n-1}$ cents. For example, if $n = 2$, $s_0 = 12$ and $s_1 = 17$, then you can buy exact postage for a letter that needs 99 cents of postage (four 12 cent stamps plus three 17 cent stamps), but you can't make exact postage for one dollar.

You can solve this problem analytically using a bit of number theory, but we won't. Instead, write a python module called `stamps.py` that uses Z3 to solve the following questions:

Given three stamp values 231 cents, 234 cents, and 238 cents,

- Can you produce exact postage of \$58.23 (i.e. 5823 cents)?
- Can you produce exact postage of \$67.00 (i.e. 6700 cents)?
- Show that you can produce exact postage of \$579.17 but that this amount is not possible if you can only use two of the three types of stamps.
- What is the largest *infeasible* postage when using stamps of 231, 234, and 238 cents?

4. Two inverters are as good as three (**40 points**)

   This problem is intended to give you some experience with uninterpreted functions. Let's say that you have a collection of logic gates that has an unlimitted number AND-gates and OR-gates, but only two inverters. Let's say you need to build a circuit that has three inputs, x, y, and z, and three outputs, $\neg x$, $\neg y$ and $\neg z$ where $\neg x$ is the logical negation of x, and likewise for y and z. You can do it!

   One solution is to think about it. This is likely to take a long time. The alternative that is required for this problem is to write code instead. This still requires a little bit of thinking, but not too much.

   First, note that we can use AND-gates and OR-gates to implement any *monotonic* function. $f(x, y, z)$ is monotonic, iff

   $$\forall x_1 \leq x_2,\ y_1 \leq y_2,\ z_1 \leq z_2\ .\ f(x_1, y_1, z_1) \Rightarrow f(x_2, y_2, z_2)$$

   For boolean valued arguments, we treat *False* $\leq$ *True*.

   We can let $f(x, y, z)$ be some monotonic function and let $u_1 = \neg f(x, yz)$. That's the first inverter. Note that the first inverter can't have any inputs that depend on the second inverter; so for any solution, there must be some function f that only depends on x, y, and z whose output is the input to the first inverter.

   Likewise, we can let $g(x, y, z, u_1)$ be some monontonic function and let $u_2 = \neg g(x, y, z, u_1)$. That's the second inverter. Finally, we need to find monotonic functions h_x, h_y, and h_z, such that

   $$\begin{array}{rcl} h_x(x, y, z, u_1, u_2) & = & \neg x \\ h_y(x, y, z, u_1, u_2) & = & \neg y \\ h_z(x, y, z, u_1, u_2) & = & \neg z \end{array}$$

   Use Z3 to find solutions for the functions f, g, h_x, h_y and h_z. You'll need to include constraints to force these functions to be monotonic and to get the negations of x, y, and z as the results.

   Just for fun. Now that you've shown how to build three inverters out of two, you can also use Z3 to show that it is impossible to build four inverters out of two.