CPSC 121 Quiz 3
Wednesday, 2012 July 18

[6] 1. Given the premises: $(\sim q \rightarrow \sim p) \wedge (q \rightarrow r)$, $\sim q \wedge r$, $d \rightarrow (s \wedge p)$, and $s \vee d$; prove s.

You may only use rules listed on "Dave's Awesome Sheet" (or lemmas you prove yourself on this quiz). Please make your finished proof clear and easy to read. **We suggest using the back of the previous page for your scratchwork!** We have begun the proof below.

Proof (that the conclusion reached below follows from the premises listed):

1.	$(\sim q \rightarrow \sim p) \wedge (q \rightarrow r)$	[premise]
2.	$\sim q \wedge r$	[premise]
3.	$d \rightarrow (s \wedge p)$	[premise]
4.	$s \vee d$	[premise]

Solution : Here's a polished, finished version.

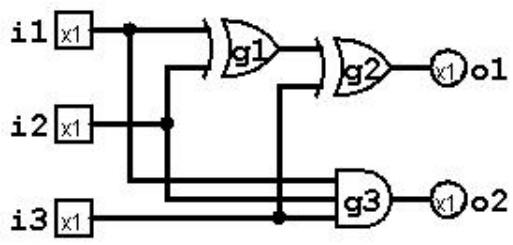
5.	$\sim q \rightarrow \sim p$	[by SPEC on 1]
6.	$\sim q$	[by SPEC on 2]
7.	$\sim p$	[by M.PON on 5, 6]
8.	$s \vee \sim p$	[by GEN on 7]
9.	$\sim(\sim s \wedge p)$	[by DM on 8]
10.	$\sim d$	[by M.TOL on 3, 9]
11.	s	[by ELIM on 4, 10]

(In our first pass, we wound up at several dead-ends, such as deriving $q \rightarrow r$ and then noticing that if we know $\sim q \wedge r$, $q \rightarrow r$ is totally useless!)

Note that it's also possible to prove this without using the first two premises. See if you can do so!

[6] 2. We can model the connections in combinational circuits with predicate logic. Let P be the set of "ports"—inputs of gates, outputs of gates, inputs of the circuit, and outputs of the circuit. Let $WireTo(a, b)$ defined for $a \in P, b \in P$ mean that a has a wire leading to b. (Notice that $WireTo$ has a *direction*. For example, we'd never have a "WireTo" relationship *from* any port *to* an input of the circuit!) Let $Input(a)$ and $Output(a)$—each defined for $a \in P$—mean that a is an input to or output from the circuit, respectively.

Consider this circuit (with gates labeled g_1, g_2, and g_3):



In this circuit, for example, $Input(i_1)$ is true because i_1 is an input to the circuit. However, g_1 (a gate) and o_1 (an output of the circuit) are not inputs; so, $Input(g_1)$ and $Input(o_1)$ are both false.

Specifically: the $Input$ predicate is true for i_1, i_2, and i_3 and false otherwise. The $Output$ predicate is true for o_1 and o_2 and false otherwise. Here are the only true $WireTo$ relationships: $WireTo(i_1, g_1)$, $WireTo(i_1, g_3)$, $WireTo(i_2, g_1)$, $WireTo(i_2, g_3)$, $WireTo(i_3, g_2)$, $WireTo(i_3, g_3)$, $WireTo(g_1, g_2)$, $WireTo(g_2, o_1)$, and $WireTo(g_3, o_2)$.

[1] (a) $Input(g_2) \rightarrow WireTo(i_1, g_2)$ is (circle one): **TRUE** **FALSE**
Remember how conditionals work!

Solution : **TRUE**. (It can only be false if $Input(g_1)$ is true but $WireTo(i_1, g_1)$ is false. Neither of these is the case.)

[2] (b) **Dis**prove the statement $\forall a \in P$, $WireTo(a, o_1)$.

Solution : This is false based on any counterexample, such as the fact that there is no wire from input i_1 to output o_1.

[3] (c) **Prove** the statement $\forall a \in P, \sim Output(a) \rightarrow \exists b \in P$, $WireTo(a, b)$.

Solution : This is true. It's trivially true for any output. So, we need only consider gates and inputs. For those, we need an example for each one showing this is true, any examples will do. For instance: all of i_1, i_2, and i_3 have wires to g_3. g_1 has a wire to g_2. g_2 and g_3 have wires to o_1 and o_2, respectively.

[8] 3. A "binary tree" is composed of "subtrees". Each subtree is either a leaf (with no children), or it has a left child and a right child, each of which is itself a subtree.

Let S be the set of all subtrees in a particular binary tree.
Let $LeftChild(p, c)$ defined for $p \in S, c \in S$ be true exactly when c is the *left* child of p.
Let $RightChild(p, c)$ defined for $p \in S, c \in S$ be true exactly when c is the *right* child of p

Now, let's further define what it means to be a binary tree.

You may always use the predicates $LeftChild$ and $RightChild$. **Furthermore, in each part, you may rely on the definitions and statements from the previous parts being correct, regardless of whether you completed them correctly!**

[2] (a) Define a predicate $Child(p, c)$ (defined for $p \in S, c \in S$) that is true exactly when c is a child of p.

Solution : We simply want to state that c is either a right or left child:

$Child(p, c) = LeftChild(p, c) \vee RightChild(p, c)$

It's tempting to add that $p \neq c$ as in:

$Child(p, c) = p \neq c \wedge (LeftChild(p, c) \vee RightChild(p, c))$

We won't take off for this, but there's a compelling reason—that matters in all problem-solving, including software design—not to do this. If $LeftChild(a, a)$ is true for

some a, why should we suddenly decide here to "overrule" that decision and make $Child(a, a)$ false? In particular: make a single important design decision (that you may wish to change) at a single place in your model. Don't entangle it throughout your model. If it's entangled in many places in your model and you later change your mind, you'll be forced to try changing it in many places... or may not even be able to!

[2] (b) Define a predicate $Root(s)$ (defined for $s \in S$) that is true when s has no parent.

Solution : The root has no parent. That is, there is no subtree such that the root is that subtree's child:

$Root(s) = \sim \exists p \in S, Child(p, s)$

[2] (c) No subtree has more than two children. State that fact in predicate logic.

Solution : That's not quite our "no more than one" idiom, but it's close. We can work from there if we understand the idiom. In this case, we're saying "it's not the case that there are three different children of a subtree". Let's start with a predicate $ThreeChildren(s)$, meaning s has at least three (different) children:

$$ThreeChildren(s) = \exists c_1 \in S, \exists c_2 \in S, \exists c_3 \in S, c_1 \neq c_2 \wedge c_1 \neq c_3 \wedge c_2 \neq c_3 \wedge Child(s, c_1) \wedge Child(s, c_2) \wedge Child(s, c_3)$$

From there, it's easy. We just want to say "no subtree has at least three children":

$\forall s \in S, \sim ThreeChildren(s)$

[2] (d) Every non-root subtree in a binary tree has a parent. State that fact in predicate logic.

Solution : We want to say something about every subtree that is not the root. That's restricting the domain of a universal to those subtrees that aren't the root. Something like:

$\forall s \in S, \sim Root(s) \rightarrow \ldots$

Now, we just want to say in the "..." that the subtree has a parent:

$\forall s \in S, \sim Root(s) \rightarrow \exists p \in S, Child(p, s)$

As we discovered in class discussing this, however, $Root(s)$ also means precisely "subtree s has no parent". Therefore, with that definition, this can be expressed as:

$\forall s \in S, \sim Root(s) \rightarrow \sim Root(s)$

Which is logically equivalent to T. So, T is a correct answer to this question. Oops!

BONUS: Earn up to 2 bonus points by doing one or more of these problems.

- A different way to think of a binary tree—closer to the typical Java version—is that it is composed (mostly) of *nodes* rather than *subtrees*. There's also one special value `null` that is not a node (or anything else) at all. So, for example, in this version every node has two children, but a node may have `null` as one or both children (and be a leaf, in the latter case). Consisder the statement "every tree has a root"; discuss whether it is true or false in the two "versions" of binary trees and then rewrite the binary tree problem with this version, explaining the differences.

  Solution : We don't include the new version, but we do ask: How must the statements change?

  Among other things, we must decide whether a `null` can be a root or not. Essentially we must decide that *either* a `null` can be the root, in which case the root can be one of these "nothing-at-all" things *or* the root must be a node, in which case we have to rewrite our definition somewhat (and maybe some of the others) with a new predicate that can test for `null`ity *and* there's one case (the empty tree) where a tree has no root at all!

  The latter seems by far the more reasonable definition.

- Part (c) of the circuit problem is a general statement that we could apply to any combinational circuit. Must it necessarily be true—or necessarily be false—for all combinational circuits? **Should** it be true or false for all combinational circuits? Justify your answers, ideally including an illustrative example or two.

  Solution : Part (c) states that every circuit "port" is either an output of the circuit or leads to some other part of the circuit. That's not true of every combinational circuit. (It's easy to construct examples, for instance, that ignore some of their inputs.) **Should** it be true? Well, what would it mean if it weren't true? Once you've thought that through, consider designing a circuit to test if a signed binary number is negative.

  Is there a similar statement we could make that we really do think *should* be true?