CPSC 121 Midterm 1
Tuesday, October 11th, 2011

[1] 1. Do you want tutorial attendance to be mandatory for you? If you answer "yes", then 1% of your course grade will be calculated with *min*$(100\%, \frac{\text{tuts attended}}{\text{tut weeks}-2})$. If you answer "no", then online quizzes will be worth 5% of your course grade rather than 4%. Either way, you get credit for this problem.

Circle one: YES NO

Solution : No solution to this question. BUT, for the rest of the exam, we indicate the learning goals with annotations like:

LGs: PropL.p1, CLE.i1

This means "Learning goals addressed were Propositional Logic pre-class goal 1 and Conditionals and Logical Equivalence in-class goal 1". We're referring to the learning goals at: http://www.ugrad.cs.ubc.ca/%7ecs121/2011W1/Homepage/goals.html.

[4] 2. Building combinational circuitry, without a story:

[2] a. Write a propositional logic expression that corresponds to the following truth table.

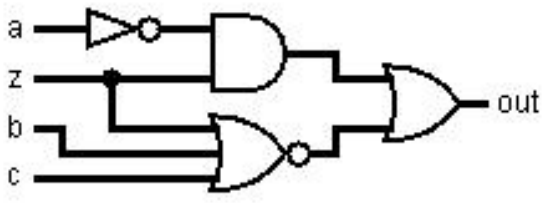
x	y	z	p
F	F	F	F
F	F	T	T
F	T	F	F
F	T	T	F
T	F	F	F
T	F	T	F
T	T	F	F
T	T	T	T

Expression:

Solution : LGs: PropL.i1
There are many possible solutions. Here's a "DNF" solution (built by "expressing the true rows" and then ORing them together): $(\sim x \wedge \sim y \wedge z) \vee (x \wedge y \wedge z)$.

[2] b. Given the following propositional logic expression, write the directly corresponding circuit: $(\sim a \wedge z) \vee \sim (b \vee c \vee z)$. Label the output *out*. Do not simplify.

Solution : LGs: PropL.p3



[3] 3. **Complete the following truth table** for the logical expression $p \wedge (h \rightarrow \sim s)$, and **then answer the question below**.

Solution : LGs: PropL.p2

p	s	h	$p \wedge (h \rightarrow \sim s)$
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	F

Is $p \wedge (h \rightarrow \sim s)$ a tautology, contradiction, or contingency?

Circle one: TAUTOLOGY CONTRADICTION CONTINGENCY

Solution : LGs: PropL.p2 (general terminology plus checking what truth tables actually mean)

It's a contingency.

[4] 4. Let s mean "your data is secure", p mean "your password is strong", and h mean "you hid your password under your keyboard".

[2] a. Translate the following from propositional logic into English: $(p \wedge s) \vee h$

Solution : LGs: PropL.p1, CLE.p1

Your password is strong and your data is secure, or you hid your password under your keyboard.

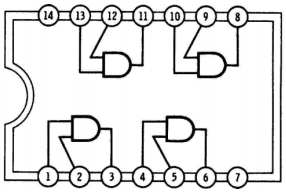
[2] b. Translate the following English statement into propositional logic. You need not translate the sentence in parentheses; it's only there to clarify the meaning of the English statement.

> If your password is weak or hidden under your keyboard, then your data is not secure. (But a strong password and not hiding your password under your keyboard don't guarantee that your data is secure.)

Solution : LGs: PropL.p1, CLE.p1

$(\sim p \vee h) \rightarrow \sim s$

[3] 5. Every practical technology for implementing digital logic has its own limitations and peculiarities that propositional logic fails to model. Consider the following chip diagram from the Magic Box Manual:



Give a shortcoming of propositional logic as a model of how difficult it is to wire a circuit using this type of chip for $(p \wedge q \wedge r \wedge s \wedge t)$ as compared to using it for $(p \wedge q \wedge r \wedge s \wedge t \wedge u)$.

Solution : LGs: labs, PropL.i2

Prop logic is an excellent model for digital logic, but there are many shortcomings.[1]

In this particular case, there's one significant difference between these two circuits that prop logic does not capture: the second one will require a second chip for the fifth AND operation.

Discussion of delay in the circuit would be OK if you made clear why this circuit would have to introduce additional delay (e.g., pointing out that a "tree" to calculate the second statement would need an extra "level" versus the first).

[7] 6. This problem focuses on the equivalence:

$(\sim p \vee s) \wedge (p \rightarrow (s \rightarrow \sim p)) \equiv \sim p$

[6] a. Prove this equivalence. Use a formal logical equivalence proof, and start your proof from the left-hand side of the equivalence.

[1] For example, NAND gates take fewer transistors (individual elements on the computer chip) to create than AND gates, yet the equivalent propositional logic expression for a NAND gate is longer than for an AND.

Solution : LGs: CLE.i1
CORRECTED (changing the rule on the ID step to ID rather than ABS):

$$
\begin{aligned}
\text{LHS} &\equiv (\sim p \vee s) \wedge (\sim p \vee (s \rightarrow \sim p)) && \text{by IMP} \\
&\equiv (\sim p \vee s) \wedge (\sim p \vee \sim s \vee \sim p) && \text{by IMP} \\
&\equiv (\sim p \vee s) \wedge (\sim p \vee \sim s) && \text{by ID} \\
&\equiv \sim p \vee (s \wedge \sim s) && \text{by DIST} \\
&\equiv \sim p \vee \text{F} && \text{by NEG} \\
&\equiv \sim p && \text{by I}
\end{aligned}
$$

[1] b. Is it possible to prove the equivalence starting from the right-hand side?

(a) Yes, and it would likely have been easier.
(b) Yes, but it's often harder to work from the simple side to the complex side.
(c) No, because a proof in that direction would not prove the logical equivalence.
(d) No, because there's no sequence of steps we can follow in that direction.

Solution : LGs: CLE.i1 (directly addressing strategies)
Yes, but it's often harder to work from the simple side.

[6] 7. Answer the following questions related to representing numbers.

[1] a. Convert the 5-bit unsigned binary number 01010 to a decimal number.

Solution Summary for all parts: LGs: NR.p1–5
10

[1] b. Convert 01001011 to hexadecimal.

Solution : 4B

[1] c. Convert the 5-bit signed binary number 11010 to a decimal number.

Solution : CORRECTED (forget to add 1!):
The bit pattern starts with 1; so, we take the 2's complement by flipping the bits (00101) and adding 1 (00110), and then convert to decimal (6), and report the negation: -6.

[1] d. Convert -9 to a 5-bit signed binary number.

Solution : We convert 9 to binary (01001) and take the 2's complement: 10111.

[1] e. Convert 4 to a 5-bit signed binary number.

Solution : 00100

[1] f. Add the 5-bit binary numbers 00101 and 10100 **or** explain why it is not possible to add them without knowing whether to interpret them as signed numbers or unsigned numbers.

Solution : 11001

[14] 8. Recall "Binary coded decimal" (BCD) from the assignment:

> BCD represents k decimal digits using $4k$ bits in groups of 4. Each group of 4 represents a single digit (0–9). So, for example, 59 would be 01011001 in BCD, a 5 (0101) followed by a 9 (1001).

[7] a. Design the logic statements for a circuit that adds 5 to a 1-digit (4-bit) BCD number $i_1 i_2 i_3 i_4$. **Assume the input value is** 4 **or less**; so, the output is 9 or less.
The rightmost bit of the output $o_4 = \sim i_4$. The second bit from the left
$o_2 = (\sim i_2 \wedge \sim i_3 \wedge \sim i_4) \vee (\sim i_2 \wedge \sim i_3 \ \wedge \ i_4) \vee (\sim i_2 \ \wedge \ i_3 \wedge \sim i_4)$.
Give one statement for each of the other two bits of output o_1 **and** o_3**.**

Solution : LGs: PropL.i1
A truth table works well and needs only five rows. We leave the i_1 column off since it's always F.

i_2	i_3	i_4	o_1	o_2	o_3	o_4
F	F	F	F	T	F	T
F	F	T	F	T	T	F
F	T	F	F	T	T	T
F	T	T	T	F	F	F
T	F	F	T	F	F	T

Rather than building the "DNF" version, we'll notice a few patterns: o_3's truth table is the same is $i_3 \oplus i_4$ for the first four rows and begins the same pattern again on the next row. (This also makes sense if you imagine adding 1 to a binary number.) o_2's truth table is the opposite of o_1's for the five rows that matter; so, we'll toss in o_2 for free! o_1 is true when i_2 is true and when i_3 and i_4 are both true.

$o_1 = (i_3 \wedge i_4) \vee i_2$
$o_2 = \sim o_1$
$o_3 = i_3 \oplus i_4$

[7] b. Design a circuit that takes a 2-digit (8-bit) BCD number $c_{11}c_{12}c_{13}c_{14}\ c_{21}c_{22}c_{23}c_{24}$ as input and produces a 2-digit (8-bit) BCD number $d_{11}d_{12}d_{13}d_{14}\ d_{21}d_{22}d_{23}d_{24}$ as output, which is the input divided by 2. Drop the remainder; for example, the input 25 produces the output 12.

In your solution you can (and should) use your circuit from the previous part plus a new one that inputs a 1-digit BCD number and outputs: (1) the result of dividing it by 2 and (2) a remainder r that is 1 when the input is odd and 0 otherwise. Assume both circuits work correctly, and represent them with the following two chip symbols:

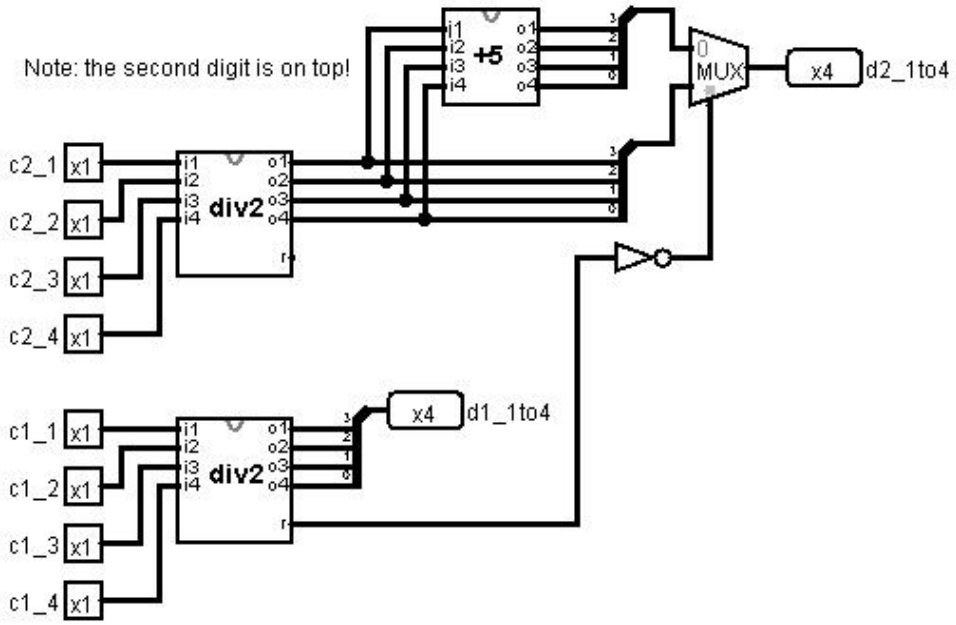
The circuits from part a (left) and b (right) as chips:

Hint: solve $\frac{28}{2}$ by hand—the easier version—and compare to solving $\frac{38}{2}$ by hand—the harder version. When did you divide digit(s) by 2 or add 5 (or maybe add 10 and then divide by 5)? If you have to choose between two operations depending on which version of the input you have, what circuit element can help?

Notes: for any multiplexer you use, indicate which of its inputs is the "0" input; use but **do not** implement (draw the "gates inside") either the `div2` or `+5` circuits here.

Solution : LGs: labs (but NOT PropL.p1 since totally different techniques)

When we divide a 2-digit BCD number by 2, we divide each digit by 2. If the first digit is even, that's the answer. If the first digit is odd, we end up with 0.5 leftover in the 10s place, which is 5. We need to add that 5 into the value in the 1s place:



(Our outputs o_1 and o_2 are 4-bit lines rather than separate 1-bit lines.)

It's clever but not quite correct to add 5 twice before dividing by 2 to get the remainder version of the right-hand digit. That's because the +5 circuit is only guaranteed to work on inputs 0–4. (In fact, an input like 9 will be too large to store in 4 bits when we add 10 to it anyway.)

[6] 9. For each of the following, apply the indicated rule to create an equivalent statement. (In some cases, you may also need to apply commutativity, associativity, or double negation. You need not write out these steps but should apply them if necessary to make the named rule apply.)

[2] a. Apply distributivity to $(p \wedge q) \vee (\sim r \wedge q)$.

Solution Summary for all: LGs: CLE.p3
$q \wedge (p \vee \sim r)$
It's also possible to distribute in the "other direction", but messier, more error-prone, and usually less useful. We gave credit for that answer but prefer the one above.

[2] b. Apply De Morgan's to $\sim p \wedge r$.

Solution : $\sim(p \vee \sim r)$

[2] c. Apply absorption to $(p \vee q \vee r) \wedge (p \vee r) \wedge (r \vee \sim s)$.

Solution : $(p \vee r) \wedge (r \vee \sim s)$

[12] 10. For each of the following, apply the indicated rule to create a statement that follows from the given statements **or** indicate that the rule does not apply. (In some cases, you may also need to apply commutativity, associativity, or double negation. You need not write out these steps.)

[3] a. Apply modus ponens to $a \to (b \wedge c)$ and a.
Write your answer or circle "does not apply": DOES NOT APPLY

Solution Summary for all: LGs: P.p2
$b \wedge c$

[3] b. Apply specialization to $(p \wedge q) \to r$.
Write your answer or circle "does not apply": DOES NOT APPLY

Solution : DOES NOT APPLY
This statement *cannot* match the form $a \wedge b$.

[3] c. Apply proof by cases to $(a \vee b) \to \sim c$ and $d \to \sim c$.
Write your answer or circle "does not apply": DOES NOT APPLY

Solution : $(a \vee b \vee d) \to \sim c$
(But you need not drop the parentheses around $(a \vee b)$ like we did.)

[3] d. Apply generalization to $p \to q$.
Write your answer or circle "does not apply": DOES NOT APPLY

Solution : $(p \to q) \vee anyoldthing$
Generalization *always* applies. We can always OR something extra onto a statement.

[4] 11. `Base64` is a scheme for encoding binary data using normal text, such as storing data in XML files (a text file format often used for web applications). In `Base64`, 64 characters—like the letters a–z, A–Z, 0–9, +, and /—represent the numbers 0–63. For example, the bit pattern 010010010110000101101111 might be represented by the characters "SWFv".

[2] a. Why choose 64 characters rather than using just the "standard" 62 characters a–z, A–Z, and 0–9?

Solution : LGs: NR.i1
$64 = 2^6$, making it easy to convert binary data into `Base64` (by converting 6-bit chunks at a time). It is also true (but not as important) that base 64 is slightly more compact than base 62 (a bit less than 1% more compact).

[2] b. Is there any data that can be encoded in normal binary that **cannot** be encoded as `Base64` data? Briefly justify your answer.

Solution : LGs: NR.i1

No. All binary data can simply be considered numbers in base 2. Any number in base 2 can be converted into a number in base 64. Any number in base 64 is representable using `Base64`.

What about something that's not a multiple of 6 bits? We can solve this with padding, e.g., adding 0s to the front until it reaches 6 bits. So, for example, 0100 becomes 000100. If it's important we preserve the original bit length, we can just include a one-character "preamble" that tells us how much padding we added: 0–5 or some similar scheme.

[6] 12. Consider the predicates Odd(x) meaning "x is an odd integer" and Prime(x) meaning "x is a prime integer" and the set of positive integers $\mathbf{Z}^+$. For each statement below, indicate whether the answer is true or false.

[2] a. $\forall x \in \mathbf{Z}^+, \text{Prime}(x) \vee \text{Odd}(x)$
Circle one: TRUE FALSE

Solution Summary for all: LGs: PredL.p2–3

False. 4 is neither prime nor odd.

[2] b. $\exists x \in \mathbf{Z}^+, \sim\text{Odd}(x) \wedge \text{Prime}(x/2)$
Circle one: TRUE FALSE

Solution : True. For example, 10 is not odd, but $10/2 = 5$ is prime.

[2] c. $\exists x \in \mathbf{Z}^+, (\sim\text{Odd}(x) \wedge x > 2) \rightarrow \text{Prime}(x)$
Circle one: TRUE FALSE

Solution : True. For example, $x = 1$ is a witness to the statement's truth. Be sure you understand why!

[15] 13. [5] a. Prove using logical equivalences that $p \rightarrow (q \rightarrow p)$ is a tautology.

Solution : LGs: CLE.i1

$$
\begin{aligned}
p \rightarrow (q \rightarrow p) &\equiv \sim p \vee (q \rightarrow p) && \text{by IMP} \\
&\equiv \sim p \vee \sim q \vee p && \text{by IMP} \\
&\equiv \text{T} \vee \sim q && \text{by NEG} \\
&\equiv \text{T} && \text{by UB}
\end{aligned}
$$

[8] b. Prove b using a formal propositional logic proof given the five numbered premises below:

Solution : LGs: P.i1

1. $(\sim p \vee q) \rightarrow p$	premise
2. $\sim r \rightarrow \sim p$	premise
3. $\sim(r \wedge \sim a)$	premise
4. $\sim a \vee b$	premise
5. $(q \vee s) \rightarrow t$	premise
6. $\sim(\sim p \vee q) \vee p$	by IMP on 1
7. $(p\wedge \sim q) \vee p$	by DM on 6
8. p	by ABS on 7
9. r	by M.TOL on 8, 2
10. $\sim r \vee a$	by DM on 3
11. a	by ELIM on 10, 9
12. b	by ELIM on 11, 4

How did we get here? We wanted b, for which we'll need to use line 4. Line 3 might give us a to get b. so, we altered its form until it looked promising. Line 2 could give us the r we needed if we had p (the negation of the negation of p). So, we played with line 1, hoping to get p and finally did.

[2] c. Assuming your proofs are correct, what do they establish? **Circle all that apply.**

(a) That $p \rightarrow (q \rightarrow p)$ is true.
(b) That $p \rightarrow (q \rightarrow p)$ is false.
(c) That b is true.
(d) That b is false.
(e) None of these.

Solution : LGs: P.i3 (but not the tightest fit)

The second proof establishes none of these facts. Instead, it shows that b follows from a set of five premises (but says nothing if the premises are not true). The first proof establishes that $p \rightarrow (q \rightarrow p)$ is true.

[15] 14. A "secure hash algorithm" takes a number as input and produces another number as output. Among other things, secure hash algorithms are used for storing passwords. Ideally, it should be hard to find an input that produces a particular output. For example, logging into an account may involve providing an input to the algorithm that matches a particular output. If someone else finds an input that produces the same output as your password, they can log into your account.

In this problem, all numbers are non-negative integers (from $\mathbf{Z}^0$), and all people are from the set P. The predicate SHA(x, y) means "the output of the secure hash algorithm run on

x is y", and Knows(p, y) means "person p knows an input to the secure hash algorithm that produces y as an output".

[5] a. Using the predicates above, write the statement "for each input to the secure hash algorithm, there is exactly one unique output" in predicate logic. As always, feel free to define and use helper predicates.

Solution Summary for all: LGs: PredL.i1
Here's a version with no "helper". There are many possible helper predicates you could use; one similar to the "MPoly" problem on the assignment would make sense.

$$\forall x \in \mathbf{Z}^0, \exists y_1 \in \mathbf{Z}^0, \text{SHA}(x, y_1) \wedge \sim \exists y_2 \in \mathbf{Z}^0, y_2 \neq y_1 \wedge \text{SHA}(x, y_2)$$

[5] b. The SHA algorithm outputs a 256-bit number. Using the predicates above, write the statement "no output of the secure hash algorithm is too large to fit in an unsigned 256-bit number" in predicate logic. Use exponentiation (write a^b to mean "a raised to the power of b") and relational operators ($<$, $\leq$, $=$, $\geq$, and $>$) if you need them.

Solution :

$$\sim \exists y \in \mathbf{Z}^0, \exists x \in \mathbf{Z}^0, \text{SHA}(x, y) \wedge y \geq 2^{256}$$

Actually, one version of SHA outputs a 256-bit number: SHA-2, 256-bit. Others output other lengths, e.g., 512 bits.

[5] c. Define a new predicate Secure(p, x) in terms of the predicates above that means "no one other than p knows any input that produces the result of running the secure hash algorithm on x".

Solution :

$$\text{Secure}(p, x) \equiv \sim \exists p_2 \in P, p_2 \neq p \wedge \exists y \in \mathbf{Z}^0, \text{Knows}(p_2, y) \wedge \text{SHA}(x, y)$$