CPSC 121 2016W2 Final Exam

May 10, 2017

Normally, the exam begins with a cover page and examination rules page. Briefly, this exam had a time limit of 90 minutes (shorter than a typical final exam of 150 minutes) and was open book/open notes. Some questions were also foreshadowed in format, particularly the questions on Predicate Logic to Proof Structure (as promised at the time of the second midterm, similar to a question on the second midterm) and Extending the Working Computer (similar to a question on an assignment and in tutorial).

All questions were sufficiently clear and unambiguous for a final exam as written; however, question 1.7 could be improved by stating clearly that a and b are lower-case English letters in both cases they are mentioned.

We also handed out copies of Dave's Awesome Reference Sheet and the Proof Strategies Tips sheet, and we strongly recommend having those on hand as you work the exam as well.

1 Short $\wedge$ (Maybe) Quick [27 marks]

1. **Draw a DFA** that meets the requirements below. **[3 marks]**

   - exactly three states: s_1, s_2, and s_3,
   - exactly one letter in its input alphabet: x,
   - the start state s_1,
   - a single accepting state s_2, and
   - **a transition function that is injective** (also known as "one-to-one")

2. Consider the following (non-contradictory) premises:

   - $b \oplus q$
   - $\sim r \rightarrow (p \wedge \sim q)$
   - $r \wedge s$
   - $(a \vee q) \rightarrow \sim s$

   Circle each of the following expressions that must also be true, given that these premises are true: **[3 marks]**

   (a) b
   (b) $\sim b$
   (c) $\sim(p \wedge \sim q)$
   (d) $\sim a \vee \sim r$
   (e) $\sim p \vee (p \wedge \sim q)$
   (f) $u \rightarrow w$

3. **Directly translate the following into propositional logic**: "The result is correct exactly when the first item had the best score but the result does **not** build on the first item's result."

   Use the definitions: $r \equiv$ the result is correct, $s \equiv$ the first item had the best score, and $b \equiv$ the result builds on the first item's result. **[2 marks]**

4. Assume that E is the set of all "items" and that we use the function $s : E \to \mathbb{N}$ to compute scores. That is, s maps each item to its score, a natural number (0, 1, 2, 3, ...). **Define a new predicate** $B(e)$ that means that item e has the **unique** largest score, in the sense that nothing else is tied for largest with e. **[3 marks]**

   $B(e) \equiv$

5. Using the same definitions, **translate the following to clear, concise English**: **[2 marks]**

   $\forall e_1 \in E, \forall e_2 \in E, |s(e_1) - s(e_2)| < 1$

6. Still using the same definitions and assuming that a and b are two different elements of E, consider the following expression. (Note the lack of absolute value bars.)

   $(\sim \exists e_1 \in E, \exists e_2 \in E, e_1 \neq e_2 \wedge s(e_1) - s(e_2) < 2) \wedge s(a) - s(b) = 4$

   (a) **Is the expression a tautology, contradiction, or contingency**? (Note: we're asking about this expression **alone**, not assuming any expressions from previous questions.) **[1 mark]**

   Circle the **best** answer: **TAUTOLOGY** **CONTRADICTION** **CONTINGENCY**

   (b) Now, **write a logically equivalent expression to the expression but with its negation moved as far inward as it can go**. **[2 marks]**

7. Knowing only that $a \in E$, $b \in E$, and $a \neq b$ and that A contains the 26 lower-case English letters ($\{a, b, c, d, \ldots, x, y, z\}$), **state clearly what we know about the cardinality (size) of each of the following**. The first one is given as an example. **Leave your answers as clear mathematical expressions** since some would be hard to calculate by hand! **[5 marks]**

   (a) **Solved example:** E. *Solution:* $|E| \geq 2$. (That is, the cardinality of E is at least two.)

   (b) $\mathcal{P}(E)$ (the power set of E)

   (c) $\mathcal{P}(A)$ (the power set of A)

   (d) $E \cap A$

   (e) $A \times A$

   (f) $A \cup A$

8. A string "hash function" maps strings to natural numbers (integers greater than or equal to zero). Imagine that you know for the hash function h that $h(\text{"hello"}) = 7$ and $h(\text{"goodbye"}) = 7$, although h is also defined for all other possible strings.

   (a) What is the domain of h? **[1 mark]**

   (b) Is h **injective** (also known as "one-to-one")? **[1 mark]**

   Circle the **best** of: **YES** **NO** **NOT ENOUGH INFO TO TELL**

   (c) Is h **surjective** (also known as "onto")? **[1 mark]**

   Circle the **best** of: **YES** **NO** **NOT ENOUGH INFO TO TELL**

   (d) Is h **bijective** (also known as a "one-to-one correspondence")? **[1 mark]**

   Circle the **best** of: **YES** **NO** **NOT ENOUGH INFO TO TELL**

9. We are proving by induction that Ryan can buy any integral number of hot dogs greater than or equal to 7 by buying a collection of 4-, 5-, 7-, and 9-hot dog packages.

   (a) Imagine that the inductive step of our proof shows that if we can buy n hot dogs with these packages, then we can buy $n + 7$ hot dogs with these packages. **Complete the portion of the inductive step of the proof after the following sentence. [1 mark]**
   "By the induction hypothesis, we can buy packages adding up to n hot dogs. So, ..."

   (b) If we were to **change** the inductive step to show instead that if we can buy n hot dogs with these packages, then we can buy $n + 5$ hot dogs with these packages, **would the number of base cases we need increase, decrease, or stay the same**? **[1 mark]**

   Circle the **best** answer: **INCREASE** **DECREASE** **STAY THE SAME**

2 Fill-in-the-Blank Induction [7 marks]

Consider the theorem that for integers $n \geq 3$, $\frac{n^n}{4} > n!$

Remember:

- n^n is n multiplied by itself n times. So, when $n = 0$, $n^n = 1$; when $n > 0$, $n^n = n * (n^{n-1})$.
- $n!$ is $n * (n-1) * (n-2) * \cdots * 3 * 2 * 1$. So, when $n = 0$, $n! = 1$; when $n > 0$, $n! = n * (n-1)!$

Assume the very useful fact that $k^{k-1} \geq (k-1)^{k-1}$ for integers $k \geq 1$.

Here are all the values smaller than 100 of each of the functions. ($\frac{5^5}{4}$ and 5! are both larger than 100.)

n	0	1	2	3	4
$\frac{n^n}{4}$	0.25	0.25	1	6.75	64
$n!$	1	1	2	6	24

Complete the following proof of the theorem by filling in the areas marked **TODO** on the right side of the page. Remember to briefly justify any step you take that is not a basic arithmetic step.

Proof by induction:

Base Case: When $n = \ldots$ **TODO: remainder of base case [2 marks]**

Inductive step: Consider an arbitrary...

TODO: specify the induction variable [1 mark]

Induction Hypothesis: Assume that...

TODO: state the IH [1 mark]

Now, we show that...

TODO: state what remains to be shown in the IS [1 mark]

TODO: finish the proof [2 marks]

Therefore, by the principle of mathematical induction, for integers $n \geq 3$, $\frac{n^n}{4} > n!$

3 Predicate Logic to Proof Structure [6 marks]

Consider the following theorem:

$$(\exists p \in T, A(p,p)) \rightarrow \forall q \in S, \exists r \in T, \exists s \in T, A(r,s) \wedge (\sim B(q,r) \vee \sim B(q,s))$$

There is **not enough information** about A, B, S, and T to prove this theorem. Instead, lay out as much of the structure of a direct proof of this theorem as you can. **Whenever you choose a specific value for a variable, specify what (if anything) this choice can depend on and any important constraints (limitations) on that variable.**

Please list each distinct step after its own bullet point so it's easy for us to follow your proof structure! (Our solution has six bullet points, as shown.)

Proof:

-
-
-
-
-
-

4 Proof by Contrapositive [6 marks]

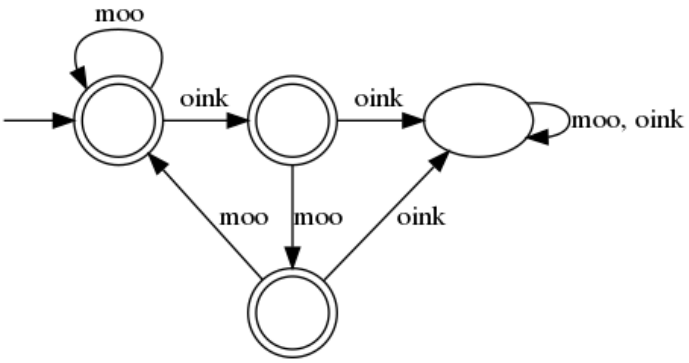
Prove the following theorem using as a central step a **proof by contrapositive**:

For any integer x, if $3x^2 + 2x - 7$ is odd, then x is even.

Be sure that you clearly and appropriately address each piece of the theorem in your proof. Substantial partial credit is available for a correctly structured (but incomplete) proof.

5 DFAs and Regexes [7 marks]

1. The following DFA accepts a certain pattern of moos and oinks:



Describe the pattern accepted by this DFA, using plain English, in no more than two sentences. **[3 marks]**

2. Write a regular expression that accepts the same sequences of oinks and moos as the DFA. Use **O** to represent an oink, and **M** to represent a moo. Otherwise, use only the regular expression language provided at the bottom of page 3 of the reference sheet. **[3 marks]**

3. Moos and oinks are animal sounds and not numbers or letters. Given this, would it be possible to translate this DFA to a circuit? **[1 mark]**

   Circle the **best** answer: **POSSIBLE** **NOT POSSIBLE**

6 Extending the Working Computer [6 marks]

EVERYTHING on this page down to the words "QUESTION BEGINS HERE" is identical to the similar tutorial/assignment problem.

A friend suggests that we should have a "subtract a constant" (or "immediate" in working computer terms) instruction. They propose that the constant can be 1 byte long or 2 bytes long and that the instruction includes one register from which we subtract the constant.

So, for example:

```
subi %ebx, $7
```

would subtract 7 from the value in register `%ebx` and update `%ebx` with the resulting value.

Your friend suggests a `subi` to subtract a 2-byte constant would look like:

C	0	rA	F	V	V	V	V

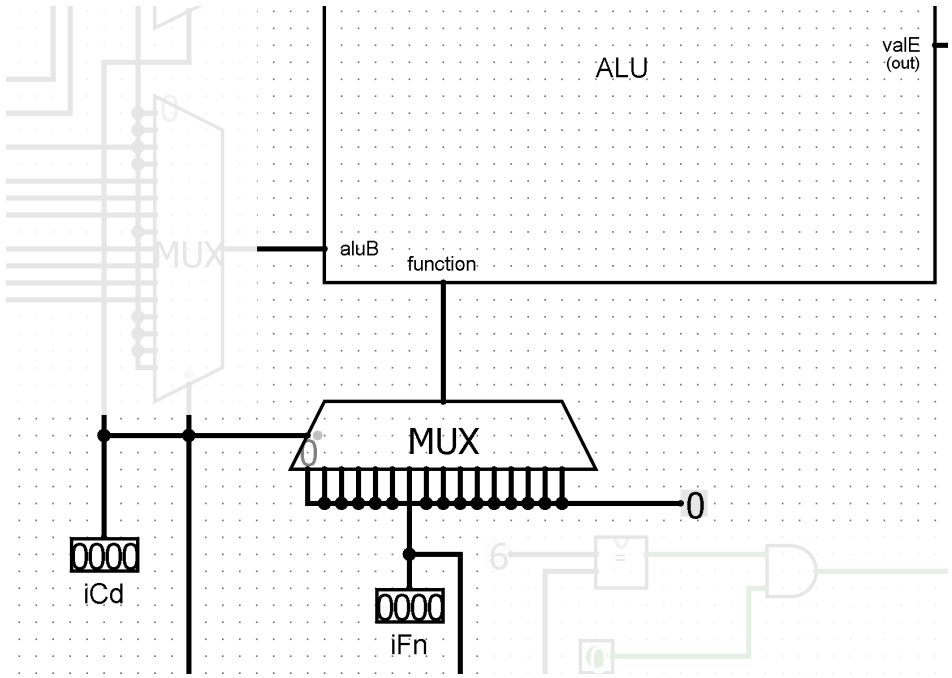
where C is a hexadecimal number (12 in decimal or binary 1100) and is the opcode for the new instruction. 0 is hexadecimal 0 and is the function code, which is ignored for this instruction, as for most instructions. rA is the register to subtract a constant from. So, if we wanted `%ebx`, it would be a 3. F is a hexadecimal number (decimal 15 or binary 1111) indicating that we do not want a second register. The 4 V's are space for the 4 hexadecimal digits representing the 2-byte long constant value to subtract, interpreted as a signed binary number.

The friend says the 1-byte version should look like:

C	0	rA	F	V	V

QUESTION BEGINS HERE

Now, consider the following portion of the "execute" stage of the computer, where the unfaded parts focus on computing the ALU's *function* (the operation the ALU is to perform). Note: when the function is 0, the ALU performs addition; when the function is 1, the ALU performs subtraction; for other functions, it performs other operations.



The input `iCd` in this image is the opcode (the first hexadecimal digit) of the current instruction. The input `iFn` is the function code (the second hexadecimal digit) of the current instruction.

1. We plan to support **both versions** of the `subi` instruction by keeping the opcode of the 2-byte version of `subi` as C but changing the opcode of the 1-byte version of `subi` from C to D.

   Use the image above to explain how you would change this portion of the circuit to implement this plan. [3 marks]

2. **Design a circuit** that takes as input the 4-bit `iCd` $i_0 i_1 i_2 i_3$ and produces true as output if the `iCd` is C or D and false otherwise (i.e., determines if the opcode is for one of the `subi` instructions). Use **only** inverters (NOTs) and 2-, 3-, or 4-input AND and OR gates in your circuit. **[3 marks]**

   (Note: when the `iCd` is C, $i_0 i_1 i_2 i_3 = 1100$.)

The rest of the page is intentionally blank.
If you write answers below, CLEARLY indicate here what question they belong with AND on that problem's page that you have answers here.

7 Induction on Algorithms [6 marks]

The following procedure (i.e., algorithm or function) adds the numbers in a list named A in a strange way as part of an efficient parallel sorting algorithm. Its second parameter c is a positive integer. It calls a helper function HELPER that works correctly, but for which we do not provide code here.

procedure SPLITANDSUM(A, c)
  if A has c or fewer elements **then**
    return HELPER(A)
  else
    Let n be the length of A
    Let A_l be the left half of A, i.e., elements with indexes i such that $0 \leq i < \lfloor \frac{n}{2} \rfloor$
    Let A_r be the right half of A, i.e., elements with indexes i such that $\lfloor \frac{n}{2} \rfloor \leq i < n$
    return SPLITANDSUM(A_l, c) + SPLITANDSUM(A_r, c)
  end if
end procedure

Imagine that we want to prove some new property $P(n)$ that describes how SPLITANDSUM operates on a list of length n. We have **already separately proven** that $P(n)$ is true for all natural numbers $n \geq 0$ for calls to HELPER.

Give the best answers possible to the following questions about a proof by induction that $P(n)$ holds for all natural numbers $n \geq 0$ when calling SPLITANDSUM, **without knowing what P is**.

1. For what value(s) of n will we write base case(s)? **[2 marks]**

2. How many times will we invoke the induction hypothesis (IH) in the proof? **[1 mark]**

3. Assuming that in the inductive step we want to establish $P(n)$ for $n = k$ (where k is an appropriate natural number), for what specific value(s) of n will we invoke the IH in the proof? **[2 marks]**

4. For this algorithm to work correctly, what should the function HELPER do with A? **[1 mark]**

The rest of the page is intentionally blank.
If you write answers below, CLEARLY indicate here what question they belong with AND on that problem's page that you have answers here.

The rest of the page is intentionally blank.
If you write answers below, CLEARLY indicate here what question they belong with AND on that problem's page that you have answers here.