Computer Science 111: Midterm Exam, Spring 2001

Prof. David Dobkin
Prof. Amit Sahai

Instructions:

1. Relax: This is not an exam in which you need to write a lot or regurgitate a lot of information. You have a full 80 minutes to take this exam.

2. There are 5 problems on 8 pages (pp. 2-9). You will need to do Problem 1 in order to do Problems 2 and 3, but otherwise the problems are independent and can be done in any order. We expect that Problem 3 will take the longest for you to complete. Some parts of problems are marked with a star – these problems are a bit harder than the rest. You may want to skip these at first and come back to them later.

3. As you read the exam questions, you will have a chance to learn a little about DNA and biotechnology. **However:** An understanding of biological concepts is not necessary to achieve a perfect score on this exam. Read the questions carefully, and they will indicate precisely what you must do.

4. This midterm is open book / open notes. You may not discuss the problems with anyone, of course, until after turning in your exam. Please write your name on every page.

5. Write out and sign the Honor Code pledge before turning in the test.

   "I pledge my honor that I have not violated the Honor Code during this examination."

Background:

Computing has applications to countless technologies, which touch our lives in many ways. One of the most exciting technological devlopments in recent years has been the emergence of biotechnology and genetic engineering, which hold tremendous promise for the future of medicine.

In this exam, we will take a look at the basic stuff of life – DNA – and think a little about how to make machines that deal with DNA.

Strands of DNA can be thought of as strings of beads, with 4 different kinds of beads possible, called bases: Adenine (A), Guanine (G), Thymine (T), and Cytosine (C).

A typical short strand of DNA might consist of the following sequence of bases: Adenine-Thymine-Cytosine-Adenine, which we could abbreviate as ATCA. Such a sequence is read in order from left to right.

(You may wonder how it is possible biologically for the body to "know" which way to read a DNA strand. Nature accomplishes this goal through some clever "hacks" that are very similar to some techniques in Computer Science. Talk to Prof. Sahai after the exam if you want to learn more.)

Problem 1 (Representation). We want to be able to represent strands of DNA in our compters. To do this, we simply need to pick an encoding of the four bases.

[**Part A**] Write down an encoding of the bases – Adenine (A), Guanine (G), Thymine (T), and Cytosine (C) – using bits.

[**Part B**] Does your encoding from Part A use as few bits as possible? If not, write down an encoding which uses as few bits as possible.

[**Part C***] Suppose that we wanted to represent whole numbers using strands of DNA. (This has actually been done and used to solve complex mathematical problems.)

Propose the most efficient method for doing this that you can, and describe it as clearly as possible. (There is an answer that takes only a few lines to describe given what we have already learned in class.)

Problem 2 (Circuits). In 1953, Watson and Crick discovered that strands of DNA actually come in pairs, and link together in a structure called the "double helix." Each base has a complimentary base with which it links: Adenine (A) links with Thymine (T) and vice-versa; Guanine (G) links with Cytosine (C) and vice-versa.

(By linking with complimentary bases and forming a double-helix, DNA becomes much more sturdy and resistant to external forces. There are many other benefits to this linking, as well.)

[**Part A**] Write down a multiple-output truth-table corresponding to complementation of bases in DNA, using the representation you chose in Problem 1B:

i.e. When given as input the encoding of Adenine (A), the output should be the encoding of Thymine (T). When given as input the encoding of Thymine (T), the output should be the encoding of Adenine (A). Similarly for Guanine (G) and Cytosine (C).

[**Part B**] Use the universal method to construct a circuit for each output of your truth table.

(Problem 2 continues on next page.)

Problem 2: **[Part C*]** Can you think of an encoding of the bases which lets you make your circuits for part [B] as simple as possible? If so, write down the new representation and new circuits. If you already picked such a representation and found the simplest circuits possible in Part [B] (and they are very simple indeed), state that you have already solved this in Part [B]).

Problem 3 (State Machines). Recently, scientists have begun building tiny machines – called nano-robots (or nanobots) – to manipulate and process DNA. In this problem, we will design our own nanobot to process DNA!

In all living beings that we know of, strands of DNA encode "instructions" for building different proteins. Proteins are chains of basic units called amino acids. Just like we used sequences of 0's and 1's to encode different instructions for our computer, in living things, sequences of Adenine (A), Thymine (T), Guanine (G), and Cytosine (C) are used to encode different amino acids.

You have been hired by a mad scientist named Dr. John "Jurassic" Johanson who wants to use Frog DNA to make giant vicious monsters in his bid to take over the world. To this end, he needs to identify all the places in a strand of DNA which could correspond to the amino acid Lysine. He tells you that Lysine is encoded by the DNA sequence Adenine-Adenine-Adenine (AAA).

Dr. J wants you to build the state machine for a nanobot designed to help him find such sequences. The nanobot moves along a DNA strand, reaching a new base in each clock cycle. Thus, once every clock cycle, the nanobot reaches a new base in the DNA strand and senses what type of base it is. This information is given to the state machine as input. The nanobot should stop as soon as it finds the sequence AAA (three Adenines in a row). **You may assume that the DNA strand always contains the sequence AAA somewhere.**

Inputs: The nanobot state machine should have as many inputs as necessary to provide an encoding of the current base (Adenine, Thymine, Guanine, or Cytosine) that the nanobot senses. (You can assume that the number of inputs to the state machine is equal to the number of bits you used in Problem 1, Part B. You can further assume that the input is encoded according to the encoding you specified in Problem 1, Part B.)

Outputs: The nanobot should only have one output, which corresponds to a Stop signal. As soon as the nanobot has detected the sequence AAA, it should enter a special "Stop" state and output 1, and stay in this state forever.

Example: If the nanobot was given the sequence ATGCAGAATAAA... it should stop after sensing the last "A", but not before.

(Rest of Problem 3 is on next 2 pages.)

Problem 3: [**Part A**] Draw the state diagram describing a State Machine for the nanobot. Clearly label the arrows between states with the what the inputs need to be in order to change from one state to the other. Don't forget to include arrows from a state back to itself if it should stay in the same state. Make sure your diagram is complete.

[**Part B**] How many bits do you need to represent your states? Pick a representation, and label your states above with their binary representations.

[**Part C**] Write down JUST 4 ROWS of the multiple-output truth table corresponding to the next state and output, given the current state and inputs to the state machine. You don't need to write down the whole truth table – just fill in 4 rows corresponding to 4 of the arrows in your state diagram from Part A.

Problem 3: [**Part D**] Finally, draw the overall final structure of the circuit for the nanobot state machine you've designed. You don't need to draw every gate, just give the overall design, using memory and the universal method as black boxes (as in Problem Set 3).

[**Part E***] Dr. J realizes that in DNA, Lysine can also be encoded by the sequence AAG. He wants you to design another nanobot which stops after seeing AAG. (It should NOT stop for any other sequence, including AAA.) You need only do [Part A] over again for this. Note that this requires a bit more care than it might appear. Think of what your machine does on the input sequence AAAG. (It should stop after sensing the last G.) Again, you may assume that the DNA strand always contains AAG somewhere.

P.S. Nanobots as simple as the ones you just helped design will play an important part in state-of-the-art biotechnology over the next few years!

Problem 4 (Numeracy). Nanobots have to be very small. Suppose that all the circuitry for a nanobot has to fit in a 10 micron by 10 micron square. Assume that a transistor is about as big as a 0.1 micron by 0.1 micron square. (Note: 1 micron = 1 millionth of a meter.)

[**Part A**] What is the absolute largest number of transistors that you could put in the nanobot?

[**Part B**] If you were using the universal method to build a logic circuit with one output bit, what is a rough estimate of how many inputs bits you could have in your truth table and still be able to build the circuit for the truth table, using only as many transistors as you computed in Part A? For this estimate, you can pretend that AND and OR gates can be constructed using only a single transistor per gate. You may also assume that the truth table has output 1 for almost all settings of the inputs. Show your reasoning.

Problem 5 (Machine Language). Dr. J decides that he will need to write some machine language programs on his TOY computer to help him with his plans to take over the world. He writes the following small program, which he intends to use later as part of a larger program. He tells you that the "inputs" to the program are the values stored in registers R1 and R2 when the program starts. The "output" of the program is the value stored in R3.

10: Load R0 ← 0000 (hex)

11: Sub R3 ← R2 - R1
12: Jump to 15 if (R3>0)

13: Add R3 ← R1 + R0
14: Jump to 16

15: Add R3 ← R2 + R0
16: halt

[**Part A**] Suppose R1=000A (hex) and R2=0002 (hex). What value will be in R3 when the program halts?

[**Part B**] Suppose R1=001C (hex) and R2=0032 (hex). What value will be in R3 when the program halts?

[**Part C**] Describe in words what Dr. J's program accomplishes.

Congratulations!

You have finished the Spring 2001 Computer Science 111 midterm!
As a reward, here is a joke for you to enjoy:

A young teacher asks her 3rd grade students to give her an example of an amphibian. Her whole class replies, "A frog!"

She is happy, and asks them, "Can any of you give me *another* example of an amphibian?"

Her students answer, "Another frog!!"

Have a great Spring Break!