CS/Math 240: Intro to Discrete Math 05/11/2011

Solutions to the Final Exam

Instructor: Dieter van Melkebeek

Problem 1

Part a

Let D be the set of all people.

From the definition of R we see that $(x, y) \in R^{-1}$ if and only if "x is a biological child of y". Then $(x, z) \in R \circ R^{-1}$ if and only if there is some y such that x is the biological child of y (i.e., if $(x, y) \in R^{-1}$), and y is the biological mother of z. In that case x and z have the same mother, so they are siblings.

Since everybody has a biological mother, every $x \in D$ is actually related to itself, too. Just let y be the biological mother of x and note that $(x, y) \in R^{-1}$ as well as $(y, x) \in R$.

The relation is symmetric. In fact, any relation of the form $R \circ R^{-1}$ is symmetric (see part (b)).

Finally, the relation is transitive. Any pair of siblings has the same biological mother, so if x and z are siblings, and z and w are also siblings, it follows that x and w are siblings too. More formally, say $(x, z), (z, w) \in R \circ R^{-1}$. Since the relation R^{-1} is injective and our hypothesis implies that x and z have the same mother, as well as z and w have the same mother, we have $(y, x), (y, z), (y, w) \in R$. Then $(x, y) \in R^{-1}$ and $(y, w) \in R$, and we see that $(x, w) \in R \circ R^{-1}$.

The equivalence classes are the sets of people who have the same mother. That is, we have a class $C_y = \{x \mid (y, x) \in R\}$ for every mother $y \in D$.

Part b

The relation $R \circ R^{-1}$ is not always reflexive. Consider the empty relation, and pick any $x \in D$. Since R is empty, there is no $y \in D$ such that $(x, y) \in R$, which means that x cannot be related to itself.

The relation $R \circ R^{-1}$ is always symmetric. Suppose $(x, z) \in R \circ R^{-1}$. That means that there is some $y \in D$ such that $(x, y) \in R^{-1}$ and $(y, z) \in R$. But then $(z, y) \in R^{-1}$ and $(y, x) \in R$, so $(z, x) \in R \circ R^{-1}$.

The relation $R \circ R^{-1}$ is not necessarily transitive. For example, suppose we have $D = \{a, b, c, d, e\}$ and $R = \{(b, a), (b, c), (d, c), (d, e)\}$. Then note $(a, b) \in R^{-1}$, so $(a, c) \in R \circ R^{-1}$. Also, $(c, d) \in R^{-1}$, so $(c, e) \in R \circ R^{-1}$. But (a, e) are not related by $R \circ R^{-1}$ because in R^{-1}, a is only related to b, and $(b, e) \notin R$.

Problem 2

Let v be a vertex of odd degree and consider the connected component $C = (V, E)$ that it belongs to. We know that for any graph, $\sum_{v \in V} \deg(v) = 2|E|$, and we derived as a corollary that every graph has an even number of vertices of odd degree. Since v is a vertex of odd degree in C, there are at least 2 vertices of odd degree in C. Let $u \neq v$ be another vertex of odd degree in C. Since C is connected, there is a path from v to u.

Problem 3

The first problem appears in the application of Euler's formula. This only holds when the graph is connected, but the graph for this problem is not connected.

Second, it is not true that all simple cycles in the graph have length 4. Going around the boundary of the 4-vertex subgraph that looks like K_4 minus one edge yields a simple cycle of length 4. Thus, the claim that every face has 3 edges on its border is incorrect.

There is another reason why the latter claim is incorrect, as the fact that the border of a face forms a simple cycle does not hold for the outer face of a disconnected graph. Indeed, in this case the outer face of the graph has 10 edges on its border (all edges except for the diagonal one in the four-vertex subgraph), whereas the largest simple cycle has length 4.

Finally, the implication $|V| - \frac{1}{3}|E| = 2 \Rightarrow |E| > 3|V|$ is incorrect. For example, if $|V| = 3$ and $|E| = 3$, this implication does not hold. A graph with $|V| = |E| = 3$ is for example the graph K_3.

Problem 4

Part a

For the graph on 4 vertices without edges, we can pick any color for each vertex. Since there are k colors, the number of colorings is k^4.

Part b

We have k choices for the color of vertex 1. Given that color, we have $k-1$ choices for the color of vertex 2, then $k-2$ choices for the color of vertex 3, and finally $k-3$ choices for the color of vertex 4. The generalized product rule applies and tells us that the total number of colorings is $k(k-1)(k-2)(k-3) = k!/(k-4)!$.

Part c

Since vertex 5 is connected to all vertices, it is quite convenient to pick its color first. We have k options for this color. After the vertex 5 is colored, we can treat the subgraph with vertices 1, 4, 5 as K_3, where one of its vertices, 5, is already colored. Now we have $k-1$ options for the color of vertex 1, and since that vertex is connected to both 1 and 5, we get $k-2$ options for the color of 4. Similarly, we get $k-1$ options for the color of 2 and $k-2$ options for the color of 3. It follows by the generalized product rule that we have $k(k-1)^2(k-2)^2$ colorings.

Part d

First let's focus on the subgraph with vertices 1, 2, 4. This graph is K_3, so there are $k(k-1)(k-2)$ ways to color its vertices. Now vertex 3 must have a color that differs from colors of 2 and 4. Those two colors are different, so we have $k-2$ choices for the color of 3. Hence, the total number of colorings is $k(k-1)(k-2)^2$.

Problem 5

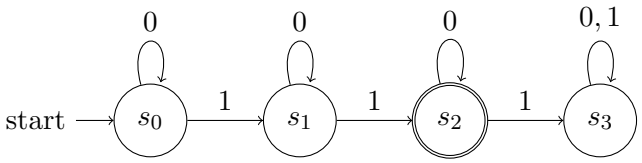
Part a

We construct an NFA for this language using two NFAs for simpler languages: (i) the language of strings that have exactly two ones, and (ii) the language of strings that have an even number of zeros.

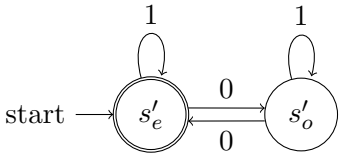
To construct an NFA that accepts strings containing exactly two ones, we start with the DFA in Figure 1a. The states s_0, s_1 and s_2 indicate that 0, 1 and 2 ones have been read so far, respectively. After reading a third one, we go to the garbage state s_3 and stay there for the rest of the computation. Now the key observation here is that since we are constructing an NFA, we can omit a transition from s_2 to the garbage state. If our NFA reads a third 1, it has nowhere to go from s_2, and, therefore, rejects right away, which is correct behavior.

We constructed an automaton accepting strings that have an odd number of ones in lecture, and the construction of an automaton that accepts strings with an even number of zeros looks almost the same. We show it in Figure 1b.

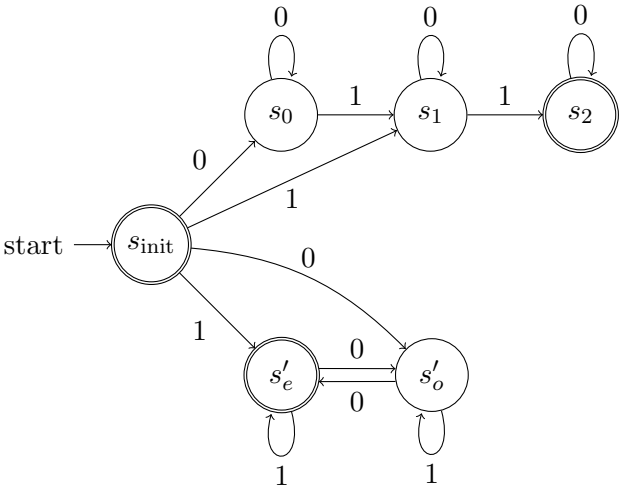
Finally, we add an initial state s_{init}. From there, we have transitions into the NFAs described in the previous two paragraphs. In particular, we copy the transitions from states s_0 and s'_e so that



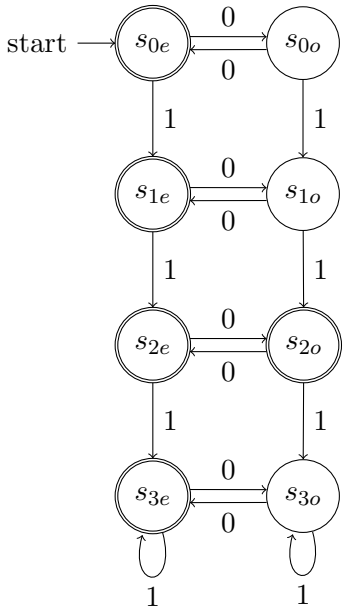
(a) DFA M_1 for strings with exactly two ones.



(b) DFA M_2 for strings with an even number of zeros.



(c) NFA for part (a)



(d) DFA for part (b)

Figure 1: Automata for problem 5.

the same transitions can be made from s_{init}. We have seen an idea like this in the constructions of NFAs for the concatenation of two languages and for the Kleene closure of a language. The final automaton is in Figure 1c. Since the NFA can correctly guess whether the string has two ones or an even number of zeros, it can correctly choose whether to go to a state of the modified machine M_1 or to a state of M_2 after reading the first input symbol. On the other hand, if a string is not in the language, it does not matter whose machine's state our automaton goes to after reading the first input symbol because both M_1 and M_2 are going to reject that string.

Part b

Our language is really a union of two languages. We can use the construction from class to design a DFA for this language. The result is in Figure 1d.

Part c

We saw a regular expression for a string containing an even number of occurrences of some symbol in lecture as well as on the homework. For an even number of occurrences of 0 in a binary string, we use the regular expression $(1^*01^*0)^*1^*$.

A string with exactly two ones may start with some number of zeros, then contain a one, then have some more zeros, another one, and end with another sequence of zeros. One regular expression for such a string is $0^*10^*10^*$.

To get a regular expression for our language, we just take the union of the two regular expressions we constructed earlier, and obtain

$$(1^*01^*0)^*1^* \cup 0^*10^*10^*.$$

Problem 6

Part a

We are going to count the set S of binary strings of length n with an even number of ones in two different ways. For the rest of this problem, we assume our strings are binary without saying so.

We can pick $2k$ out of n positions in a string of length n for $k \in \{0, \ldots, \lfloor n/2 \rfloor\}$. Put ones in those positions, and put zeros everywhere else. This process yields a string of length n with an even number of ones. We can pick $2k$ positions for the ones in $\binom{n}{2k}$ ways, and we can pick $2k$ ones for $k \in \{0, \ldots, \lfloor n/2 \rfloor\}$. Since the set of strings with $2i$ ones and the set of strings with $2j$ ones are disjoint whenever $i \neq j$, we can use the sum rule to count the number of strings with an even number of ones using $\sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k}$.

Now let T be the set of strings of length $n-1$, and let $p : \{0,1\}^{n-1} \to \{0,1\}$ be the parity function. That is, $p(x) = 1$ if x contains an odd number of ones, and is 0 otherwise. Let $f : T \to S$ be defined by $f(x) = xp(x)$, i.e., x concatenated with the parity of x. This function is well-defined. If x has an odd number of ones, $p(x) = 1$, so the number of ones in $xp(x)$ is one more than the number of ones in x, which is even. If x has an even number of ones, $p(x) = 0$, so concatenating x and $p(x)$ produces a string with an even number of ones.

Observe that f is total since we defined it for every string in T. Also, f is injective. Suppose $f(x) = f(y)$. This means that the first $n-1$ bits of $f(x)$ and $f(y)$ are the same, and those bits are x and y, respectively, so $x = y$. Finally, f is surjective. Let z be a string of length n with an even number of ones, and let x be the first $n-1$ bits of z. If the last bit of z is 1, x has an odd

number of ones, so $p(x) = 1$ and $xp(x) = z$. If the last bit of z is 0, x has an even number of ones, $p(x) = 0$, and $xp(x) = z$. Thus, f is a bijection from the set of strings of length $n-1$ to the set of strings of length n with an even number of ones. Since there are 2^{n-1} strings of length $n-1$, there are 2^{n-1} strings of length n with an even number of ones by the bijection rule.

Since both of our approaches counted the number of elements in the set S, we have $2^{n-1} = |S| = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k}$, which is what we wanted.

Part b

The binomial theorem states that

$$(x+y)^n = \sum_{\ell=1}^{n} \binom{n}{\ell} x^\ell y^{n-\ell}. \tag{1}$$

Plugging in $(x, y) = (1, 1)$ and $(x, y) = (-1, 1)$, respectively, into (1) yields the following two equations:

$$2^n = \sum_{\ell=0}^{n} \binom{n}{\ell} 1^\ell 1^{n-\ell} \tag{2}$$

$$0 = \sum_{\ell=0}^{n} \binom{n}{\ell} (-1)^\ell 1^{n-\ell} \tag{3}$$

Now we can add (2) and (3) to get

$$2^n = \sum_{\ell=0}^{n} \binom{n}{\ell} (1^\ell 1^{n-\ell} + (-1)^\ell 1^{n-\ell}) = \sum_{k=0}^{n} \binom{n}{k} (1^\ell + (-1)^\ell) 1^{n-\ell}. \tag{4}$$

Since $1^\ell + (-1)^\ell$ is 2 if ℓ is even and 0 if ℓ is odd, and since the largest term with an even value of ℓ in (4) is $2\lfloor n/2 \rfloor$, we can rewrite (4) as

$$2^n = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k} 2,$$

and divide by 2 to get

$$2^{n-1} = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k}.$$