CSC238:Discrete Mathematics for Computer Science – Spring 2003

Solutions for Assignment 4.

1. For each of the following assertions, state whether it is true or false, and justify your answer. You can use any means of justification you like, but if you use the logical equivalences stated in the course notes then specify which ones. Here $A(x)$ and $B(x)$ are unary predicates in the first order language $\mathcal{L}$.

   (a) $\forall x(A(x) \to B(x))$ logically implies $\exists x(A(x) \wedge B(x))$.

   (b) $\exists x(A(x) \to \neg B(x))$ is logically equivalent to $\neg \forall x(A(x) \wedge B(x))$.

   (c) $\exists x A(x) \wedge \exists x \neg A(x)$ is logically equivalent to $\exists x(A(x) \wedge \neg A(x))$.

   (d) $\exists x A(x) \vee \exists x \neg A(x)$ is logically equivalent to $\exists x(A(x) \vee \neg A(x))$.

   Answers:

   (a) $\forall x(A(x) \to B(x))$ **does not** logically imply $\exists x(A(x) \wedge B(x))$.

   Proof. Suppose S is a structure for $\mathcal{L}$ with domain D such that the predicates $A(x)$ and $B(x)$ are associated with relations $A^S = \emptyset$ and $B^S \subseteq D$. Therefore the antecedent $A(x)$ is false for all x and, as a result, $\forall x(A(x) \to B(x))$ is true. Also, since $\exists x A(x)$ is false, $\exists x(A(x) \wedge B(x))$ is also false for this structure. Therefore S is a structure for which $\forall x(A(x) \to B(x))$ is true and $\exists x(A(x) \wedge B(x))$ is false, and we conclude $\forall x(A(x) \to B(x))$ cannot logically imply $\exists x(A(x) \wedge B(x))$.

   (b) True. A proof is as follows:

   $$\begin{array}{llll} \exists x(A(x) \to \neg B(x)) & \text{LEQV} & \exists x(\neg A(x) \vee \neg B(x)) & \text{by } \to \text{ rule,} \\ & \text{LEQV} & \exists x \neg(A(x) \wedge B(x)) & \text{by De Morgan's law,} \\ & \text{LEQV} & \neg \forall x(A(x) \wedge B(x)) & \text{by Rule I, p.186,.} \end{array}$$

   (c) $\exists x A(x) \wedge \exists x \neg A(x)$ is **not** logically equivalent to $\exists x(A(x) \wedge \neg A(x))$.

   Proof. Let S be a structure for $\mathcal{L}$ with domain $D = \{a, n\}$, and suppose $A(x)$ is associated with the relation $A^S = \{a\}$. Then $A(x)$ is true for valuation σ_a^x and false for valuation σ_n^x . Since the two predicates $A(x)$ in the formula $\exists x A(x) \wedge \exists x \neg A(x)$ appear in different scopes, these two valuations show that this formula is true for this structure S. However $A(x) \wedge \neg A(x)$ is unsatifisfiable, and hence is false for any interpretation. Therefore $\exists x(A(x) \wedge \neg A(x))$ is false. Thus the two formula in question have different truth values for this structure S, hence they cannot be logically equivalent.

   (d) $\exists x A(x) \vee \exists x \neg A(x)$ is logically equivalent to $\exists x(A(x) \vee \neg A(x))$.

   Proof. Both formulas are valid. Note there are two separate scopes for x in the first formula, and that it is logically equivalent to $\exists x A(x) \vee \exists y \neg A(y)$. Let S be any structure, with domain D. Then either $A^S = \emptyset$ and $\exists y \neg A(y)$ must be true (since $D \neq \emptyset$), or $A^S \neq \emptyset$ and $\exists x A(x)$ must be true. In either case we see that $\exists x A(x) \vee \exists x \neg A(x)$ is true.

Similarly, for the second expression, the subformula $A(x) \vee \neg A(x)$ must be true for any valuation $\sigma(x)$, and hence $\exists x(A(x) \vee \neg A(x))$ must be true. Therefore both formula are true for any interpretation, and hence are logically equivalent.

2. Convert the following first order formula into a logically equivalent formula in Prenex Normal Form (PNF) which only uses the connectives $\neg$, $\wedge$ and $\vee$ (along with the quantifiers $\exists$ and $\forall$, of course). In particular, **do not use** the connective $\rightarrow$.

$$(\forall x \neg T(z,x) \rightarrow S(x,y,z)) \ \wedge \ (\neg \exists y \forall z R(y,z,x) \rightarrow \exists x U(x)).$$

Show the steps through which you obtained your PNF formula.

A derivation is as follows. For the first term:

$$\begin{array}{lll} (\forall x \neg T(z,x) \rightarrow S(x,y,z)) & \text{LEQV} & (\forall w \neg T(z,w) \rightarrow S(x,y,z)) \\ & \text{LEQV} & \exists w(\neg T(z,w) \rightarrow S(x,y,z)) \text{ by Rule IIe, p. 188,} \\ & \text{LEQV} & \exists w(\neg\neg T(z,w) \vee S(x,y,z)) \\ & \text{LEQV} & \exists w(T(z,w) \vee S(x,y,z)) \end{array}$$

For the second term:

$$\begin{array}{lll} \neg \exists y \forall z R(y,z,x) \rightarrow \exists x U(x)) & \text{LEQV} & \neg \exists t \forall u R(t,u,x) \rightarrow \exists v U(v) \\ & \text{LEQV} & \neg\neg \exists t \forall u R(t,u,x) \vee \exists v U(v) \\ & \text{LEQV} & \exists t \forall u R(t,u,x) \vee \exists v U(v) \\ & \text{LEQV} & \exists t \forall u \exists v (R(t,u,x) \vee U(v)). \end{array}$$

The orginal formula is therefore logically equivalent to the conjuction of these two terms. Since the quantifiers are all on distinct variables, they can be brought to the front of the expression. We find the original expression is logically equivalent to

$$\exists w \exists t \forall u \exists v ((T(z,w) \vee S(x,y,z)) \wedge (R(t,u,x) \vee U(v))).$$

Only the order of the quantifiers on variables t and u matters.

3. Consider a first order language $\mathcal{L}$ with predicates $S(s,n,a)$, $C(c,n)$, $E(s,c,y,m)$ and $T(n,c,y)$, and the equality relation $\approx$. Consider a structure for $\mathcal{L}$ which consists of the following relations corresponding to the predicates S, C, E, and T, respectively:

- $Student(s,n,a)$ – each student corresponds to a unique triple (s,n,a), where s is the unique student number, n the student's name, and a is the student's address (all in the form of character strings).
- $Course(c,n)$ – each course corresponds to a unique pair (c,n) where c is the unique course identifier (e.g. "CSC238") and n is the course title (e.g. "Discrete Mathematics for Computer Science").

- *Enrolled*(s, c, y, m) – a quadruple (s, c, y, m) indicates that the student with student number s was enrolled in the course c in the year y and obtained the mark m. If the course is currently being taken by the student, then the mark is recorded as an "I" for incomplete.
- *Teaches*(n, c, y) – a triple (n, c, y) indicates that a professor named n taught the course named c in the year y.

The domain D for this structure is the set of strings that form entries in this database. For example, these strings include all the student numbers, names, addresses, course names, course identifiers, years, and so on. We will also use these same strings as constants in the first order language, so a formula such as $(n \approx \text{"}Jepson\text{"})$ is true only when the valuation σ is such that $\sigma(n)$ is the string "Jepson" in the domain D.

Write first order formulas expressing each of the following queries:

(a) Find the name and address of every student who took CSC238 in 2001.

$$\exists s(\exists m E(s, \text{"}CSC238\text{"}, \text{"}2001\text{"}, m) \wedge S(s, n, a))$$

(b) Find the names of all the courses ever taught by Cook.

$$\exists y T(\text{"}Cook\text{"}, c, y)$$

(c) Find the names of all the courses taught *only* by Cook.

$$\exists y T(\text{"}Cook\text{"}, c, y) \wedge \forall n(\exists y T(n, c, y) \rightarrow (n \approx \text{"}Cook\text{"}))$$

(d) Find the names of every student who has received an "A" or an "I" in every course he/she has taken.

$$\exists s(\exists a S(s, n, a) \wedge \forall c \forall y \forall m(E(s, c, y, m) \rightarrow ((m \approx \text{"}A\text{"}) \vee (m \approx \text{"}I\text{"}))))$$

(e) Find the names of every course that has not been taken by any student who has ever enrolled in CSC238.

$$\exists t C(c, t) \wedge \forall s(\exists y \exists m E(s, c, y, m) \rightarrow \neg \exists y \exists m E(s, \text{"}CSC238\text{"}, y, m))$$

4. Consider the alphabet $\Sigma = \{0, 1\}$ and the language

$$L = \{x \in \Sigma^* : \text{01 and 10 are both substrings of } x \text{ (perhaps overlapping)}\}.$$

For example, $010 \in L$ since 01 and 10 are both substrings of 010, even though these substrings overlap on the character 1. Construct a regular expression that denotes L, and prove that it is correct.

Answer. A suitable regular expression is

$$R = (00^*11^*0 + 11^*00^*1)(0+1)^*$$

Proof. Let $s \in L$. Let $n = |s|$ and suppose k and j are the indicies for the beginning characters of the substrings 01 and 10, respectively. By the definition of L these indicies must exist, and $k \neq j$ with $1 \leq k < n$ and $1 \leq j < n$. By definition of a substring notice that we have $s_k = 0$, $s_{k+1} = 1$, and $s_j = 1$, $s_{j+1} = 0$.

Suppose $k < j$, that is the substring 01 appears before 10 in s. Since the $(j+1)^{st}$ element of s is 0, we have $s = uv0w$ where $u = s_1 \ldots s_k$, $v = s_{k+1} \ldots s_j$, $w = s_{j+2} \ldots s_n$ (if $n = j+1$ then we take this notation to mean $w = \epsilon$). Also, since the first index for the substring 01 is k it follows that $u_k = 0$. Moreover, since the first occurrence of the string 10 in s is at $j > k$, this string 10 cannot appear in u. It therefore follows that u must be all zeros. (We can prove this by contradiction. Suppose u is not all 0's. Let i be the maximum index of a 1 in u, so $u_i = 1$. Notice we already know $u_k = 0$ so $i < k$. Since i is the maximum index for which $u_i = 1$, and $i < k$, we infer $u_{i+1} = 0$. Therefore 10 is a substring of u starting at index $i < k$. This contradicts the assumption that the first index of the substring 10 is at index $j > k$. Therefore u must be all 0's.) Note that $u_k = 0$ so u is not the null string. Thus $u \in \mathcal{L}(00^*)$.

Since 01 is a substring of s with index k, it follows that $s_{k+1} = v_1 = 1$. From the fact that the substring 10 first appears at index $j > k$ it follows that $v \in \mathcal{L}(11^*)$. (This is proved by contradiction, in a similar fashion to the argument above that u must be all 0's. We omit the details.) Therefore we have shown that $uv0$ is a prefix of s and $uv0 \in \mathcal{L}(00^*11^*0)$. The suffix w of s starting at index $j+2$ (or the null string if $n = j+1$) is clearly in $\mathcal{L}((0+1)^*)$. Therefore it follows that $s \in \mathcal{L}(00^*11^*0)\mathcal{L}((0+1)^*) = \mathcal{L}(R)$.

For the second case we have $j < k$. A similar argument shows that $s = uv1w$ with $u \in \mathcal{L}(11^*)$, $v \in \mathcal{L}(00^*)$, and $w \in \mathcal{L}((0+1)^*)$. We omit the details. It again follows that $s \in \mathcal{L}(R)$.

Therefore in both cases we have shown $s \in \mathcal{L}(R)$ and, since s was an arbitrary element of L, we conclude that $L \subseteq \mathcal{L}(R)$.

To show the reverse, namely $\mathcal{L}(R) \subseteq L$, suppose $s \in \mathcal{L}(R)$. Then by the definition of regular expressions $s = xw$ with $w \in \mathcal{L}((0+1)^*)$ and either $x \in \mathcal{L}(00^*11^*0)$ or $x \in \mathcal{L}(11^*00^*1)$.

There are therefore two cases for the prefix x. In the first case, $x \in \mathcal{L}(00^*11^*0)$. Thus $x = uv0$ with $u \in \mathcal{L}(00^*)$, $v \in \mathcal{L}(11^*)$. Let $k = |u|$ and $j = |v|$. It follows that $k, j \geq 1$. Therefore $x_k x_{k+1} = u_k v_1 = 01$, and 01 must be a substring of x. Moreover, since $x = uv0$ and $|v| = j \geq 1$ we find that $v_j 0$ is a suffix of x. Since $v \in \mathcal{L}(11^*)$ it follows that $v_j = 1$ and therefore 10 is a suffix of x. Thus we have shown that 01 and 10 are both substrings of x. Since x is a prefix of s, they must also be substrings of s. Therefore $s \in L$ for the current case $x \in \mathcal{L}(00^*11^*0)$.

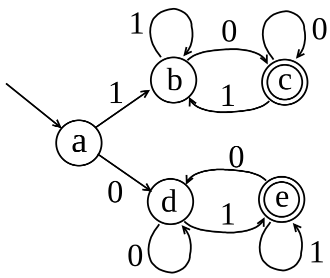
Alternatively, in the second case, $x \in \mathcal{L}(11^*00^*1)$. A similar argument to the one in the previous paragraph shows that 10 and 01 must be substrings of the prefix x and hence of the original string s. We omit the details. Therefore, in both cases, $s \in L$. Since s was an arbitrary element of $\mathcal{L}(R)$, it follows that $\mathcal{L}(R) \subseteq L$.

Since we have shown that $L \subseteq \mathcal{L}(R)$ and $\mathcal{L}(R) \subseteq L$ we conclude that $\mathcal{L}(R) = L$, as desired.

5. For each of the following languages over the alphabet $\Sigma = \{0,\ 1\}$, construct a regular expression that denotes it and construct a DFSA that accepts it. You do not need to provide proofs that your constructions are correct.

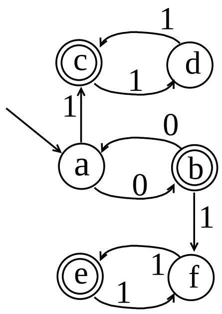
   (a) $L = \{x \in \Sigma^* \ : \ x \neq \epsilon$, and the first and last symbols of x are different$\}$,
   Answer:

$$R = (0(0+1)^*1) + (1(0+1)^*0).$$



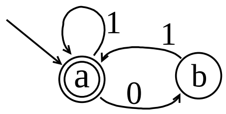
   (b) $L = \{x \in \Sigma^* \ : \ x = 0^n1^m$ with $n, m \geq 0$ and $n+m$ is odd$\}$,
   Answer:

$$R = (0(00)^*(11)^*) + ((00)^*1(11)^*).$$



   (c) $L = \{x \in \Sigma^* \ : \ $ every 0 in x (if any) is immediately followed by a 1 $\}$.
   Answer:

$$R = 1^*(011^*)^*$$



6. Consider the alphabet $\Sigma = \{0, 1\}$, and language

$$L_3 = \{x \in \Sigma^* \ : \ x = y1z \text{ with } y, z \in \Sigma^* \text{ and } z \text{ has length } |z| = 2\}.$$

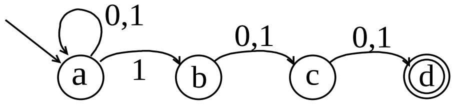
That is, L_3 is the language of binary strings of length at least 3 for which the third last character is 1.

(a) Write a regular expression which denotes L_3.

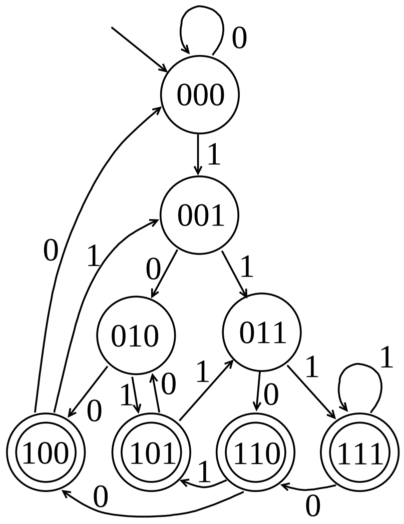
Answer:

$$R_3 = (0 + 1)^* 1(0 + 1)(0 + 1)$$

(b) Construct a NFSA with 4 states which accepts L_3.



(c) Construct a DFSA with 8 states which accepts L_3.



(d) Consider generalizing the definition of L_3 by defining L_k to be the language in Σ^* such that the k^{th} bit from the right end is 1. Show the form of a $k + 1$ state NFSA which accepts L_k for each $k \geq 1$. How many states are required for a DFSA which accepts L_k (see the hint in part c above)? Explain (but you do not need to prove anything here).

Answer:

An NFSA that accepts L_k has $k + 1$ nodes and is similar to the NFSA shown in the solution for part 6b above. The only change needed is to adjust the length of the chain of nodes after the state b. In order to accept L_k this chain should have length $k - 1$. Each of the intermediate nodes in this chain should have the same form as the node c in the solution to part 6b, with incoming and outgoing branches having the labels $0, 1$. Only the last of node in this chain is an accepting node.

A DFSA which accepts L_k must have distinct states for (at least) each of the possible suffixes of length k. To see this, suppose x is an input sequence. After processing a prefix of x, the decision of whether or not x will be accepted can depend on the precise value of each of the symbols in the suffix of length k. That is, if no more symbols occur, then the k^{th} bit from the end will determine acceptance. If only one more symbol is left in x, then the acceptance is determined by the $(k-1)^{st}$ bit from the end of the current prefix. Similarly for the symbols $k-2, \ldots, 1$ from the end of the prefix. Therefore the DFSA must have distinct states for each of these possible symbol strings of length k.

Since the alphabet has two symbols, the number of strings of length k is 2^k. Therefore, any DFSA which accepts L_k must have at least 2^k states. Note that this is exponentially larger than the number of states (namely $k+1$) required by an NFSA which accepts L_k.