University of Virginia - cs3102: Theory of Computation Spring 2010

PS1 - Comments

Problem 1: Sets and Languages. For each pair of sets, A and B answer whether they are equal, $A \subseteq B$, $B \subseteq A$, or none of these are true. Include a brief proof supporting your answer. We use $\mathcal{N}$ and $\mathcal{Z}$ as defined in Sipser: $\mathcal{N}$ is the set of natural numbers; $\mathcal{Z}$ is the set of integers.

a. $A \equiv \{\}, B \equiv \{\{\}\}$

Answer: $A \subset B$**.** A is the empty set, a set containing no elements. B is a set containing one element, which is an empty set. The empty set is a subset of every set.

b. A and B are sets such that $A \cup B = A \cap B$.

Answer: **Equal.** Proof by contradiction: assume A and B are not equal. Then, either (1) there is some element $x \in A$ such that $x \notin B$. If this is the case, then x is in $A \cup B$ but not in $A \cap B$, so this cannot be true; or (2) there is some element $y \in B$ such that $y \notin A$. If this is the case, then y is in $A \cup B$ but not in $A \cap B$, so this cannot be true. Both possibilities are false, so we have a contradiction. Thus, $A = B$.

c. $A \equiv \mathcal{N} \times \mathcal{Z}, B \equiv \{(i,j) \mid i,j \in \mathcal{N}\}$

Answer: $B \subset A$**.** $B \equiv \mathcal{N} \times \mathcal{N}$ and $A \equiv \mathcal{N} \times \mathcal{Z}$. Since $\mathcal{N} \subseteq \mathcal{Z}$, $\mathcal{N} \times \mathcal{N} \subseteq \mathcal{N} \times \mathcal{Z}$.

d. $A \equiv \{i \mid i \in \mathcal{Z} \wedge i \notin \mathcal{N}\}$
$B \equiv \{i \mid i = -k \text{ for some } k \in \mathcal{N}\}$

Answer: $B \subset A$**.** A contains 0, but B does not, so they cannot be equal. Since every negative natural number is an integer, every element of B is an element of A, so $B \subseteq A$.

e. $A \equiv$ the set of all binary relations that are not symmetric.
$B \equiv \{R \mid R \text{ is a binary relation and for every } x \text{ there exists a } y \text{ such that } xRy \neq yRx\}$

Answer: $B \subset A$**.** For a binary relation R to be symmetric, $\forall x,y : xRy \implies yRx$. This means there does not exist any x and y such that $xRy \neq yRx$. So, the set of binary relations that are not symmetric is the set of all binary relations R such that there exists at least one x and y such that $xRy \neq yRx$. Set B is the set of binary relations R such that for *every* x there exists some y such that $xRy \neq yRx$. By the definition, we know $B \subseteq A$ since the definition of B is more restrictive than the definition of A. To show they are non-equal, we need to find some relation R that is in A but not in B. Here's one:

If $x = 0$ or $y = 0$, $xRy =$ true. Otherwise, $xRy = x < y$.

For $x = 0$, there is no y such that $xRy \neq yRx$. Thus, this relation is not in B. It is, however, in A, since $3R4 \neq 4R3$.

Problem 2: Functions. Describe three ways a Java method is different from a mathematical *function* (as defined in Section 0.2).

Answer: Unlike a mathematical function which always produces the same output for the same input, a Java method may produce different outputs when it is applied multiple times to the same input. (For example, `java.util.Random.nextInt()` produces a random number every time it is called.)

Unlike a mathematical function that produces an output for all inputs in its domain, a Java method may fail to produce an output for some inputs in its domain. For example, it may fail to terminate or produce an exception.

Unlike a mathematical function which is an abstract mapping that takes no resources, a Java method takes resources (including time, energy, and space) to produce a result.

Unlike a mathematical function which has infinite precision, a Java method can only output results that can be represented on the execution platform. Thus, even simple mathematical functions like $+$ cannot be correctly implemented as Java methods for all inputs.

Problem 3: Graphs and Trees. These questions are intended to check your careful understanding and precise application of the definitions in Section 0.2.

Consider the undirected graph $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = V \times V$.

a. How long is the longest *simple path* in G?

Answer: **3.** A simple path is a path that doesn't repeat any nodes. Thus, the longest simple path in G will use each node once (e.g., $1 \longmapsto 2 \longmapsto 3 \longmapsto 4$). The length of a path is the number of edges, so this path has length 3.

b. How many subgraphs does G have?

Answer: Recall the subgraph definition: G' is a subgraph of G if the nodes of G' are a subset of the nodes of G, and the edges of G' are the edges of G on the corresponding nodes. Thus, we can create a subgraph for any subset of the nodes in G, but once the nodes are selected there is no choice what edges to include. There are $\mathcal{P}(G)$ possible subsets of G, so there are $2^4 = 16$ possible subgraphs. (Note that the empty graph and G itself do count as subgraphs of G.)

c. How many of the subgraphs of G are *trees*?

Answer: **1.** A tree is a graph that is connected and has no simple cycles. In G, every node has a self-cycle since $E = V \times V$ which includes elements like $(1,1)$. Hence, any subgraph of G that contains any nodes has a cycle and is not a tree. The only subgraph that is a tree is the empty subgraph (no nodes), so there is **1** subgraph that is a tree.

Problem 4: Roshambo. [This question was removed from this problem set.]

Problem 5: Subsets. Prove that the set subset operator ($\subseteq$) is (a) reflexive and (b) transitive, but (c) not symmetric.

Answer: We use A, B and C to represent any set.

a. Reflexive: $A \subseteq A$. By the definition of $\subseteq$, the sets are equal so every member of A is also a member of A.
b. Transitive: $A \subseteq B$ and $B \subseteq C$ implies $A \subseteq C$. Proof by contradiction: suppose $A \subseteq B$ and $B \subseteq C$ but $A \supset C$ (note that $\supset$ is the opposite of $\subseteq$). This would mean there is some element $x \in A$ such that $x \notin C$. But, $A \subseteq B$, so x must be in B. And, $B \subseteq C$, so x must be in C. This contradicts the premise, so provies that $A \subseteq C$.
c. Not Symmetric: $A \subseteq B$ does not imply $B \subseteq A$. We can prove an implication is not true by finding one counterexample: $A = \{\}, B = \{1\}$. Here, A is a subset of B, but B is not a subset of A.

Problem 6: Powerset. Prove that the size of the powerset of any set S is $2^{|S|}$. (Hint: use induction on the size of the set.)

Answer: We prove the property by using induction on the *size of the set* which is a nonnegative integer.

Basis. $S = \emptyset$. For the empty set, $|S| = 0$. The power set is $\emptyset$, which is a set of size 1. Since $2^0 = 1$, the property holds. (Note: many of your proofs selected the set of size 1 as your basis. This still works for the induction, but isn't enough to prove the property for *any* set, since the induction proof proves the property for the basis and all large sets, but $\emptyset$ is a set too.)

Induction. Prove that if the property holds for a set S of size n, it also holds for the set $S' = S \cup x$ for any $x \notin S$. The elements of the power set of S' are the elements of power set of S (since S is a subset of S', all of the subsets of S are also subsets of S') and each element of the power set of S with x added. Thus, the size of the power set of S' is *twice* the size of the power set of S: $|\mathcal{P}(S')| = 2|\mathcal{P}(S)|$. By the induction assumption, $|\mathcal{P}(S)| = 2^{|S|}$, so $|\mathcal{P}(S')| = 2 \times 2^{|S|} = 2^{|S|+1} = 2^{|S'|}$.

Problem 7: Pizza Slicing. Consider the problem of slicing a circular pizza into pieces using cuts that are straight lines all the way across the pizza. What is the maximum number of pieces that can be cut using n cuts? Support your answer with a convincing argument.

Answer:

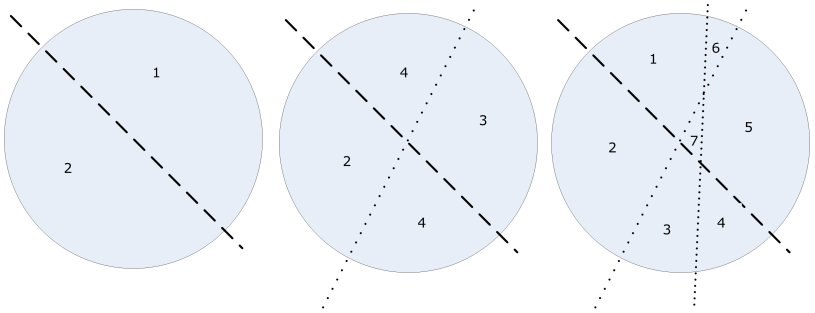


Figure 1: Three Cuts

Looking at our example in Figure 1, after 0 cuts there is 1 piece; after 1 cut, 2 pieces; after 2 cut, 4 pieces; after 3 cuts, 7 pieces. The key observation is that the number of pieces is maximized if each cut bisects all previous cuts. We can always find cuts that do this (although it gets harder to draw them), as long as none of the cuts are parallel to each other (and we can make infinitely many cuts preserving the non-parallel property). Everytime a cut is bisected, one new piece is created. So, the n^{th} cut will bisect $n-1$ previous cuts, creating $(n-1)+1=n$ new pieces (the $+1$ is for the piece the new cut creates itself). Thus, the number of pieces after n cuts is the number of pieces after $n-1$ cuts + the number of new pieces = n. This is the sum of the integers: $Pieces(n) = 1+1+2+3+4+...+n = 1+\sum_{i=1...n} i$.

Intuitively, we can see that the sum of the first n integers is $n(n+1)/2$ since we can sum the pairs of elements from the outside in: $1+n = n+1$, $2+(n-1) = n+1$, $3+(n-2) = n+1$, ... There will be $n/2$ pairs, so the sum is $n(n+1)/2$. We prove this using induction.

Basis. If $n=0$, $\sum_{i=1...n} i = 0 = 0(0+1)/2$.

Induction. Assume $\sum_{i=1...n} i = n(n+1)/2$, show $\sum_{i=1...n+1} i = (n+1)(n+1+1)/2$. From the induction assumption, we know $\sum_{i=1...n+1} i = n+1+\sum_{i=1...n}$. Using algebra, we can simplify this: $n+1+(n(n+1)/2) = n+1+(n^2+n)/2 = (n^2+3n+2)/2 = (n+1)(n+2)/2$. This proves the induction hypothesis.

(You saw a graphical proof of this result in Prof. Robins' lecture. This result was famously shown by Karl Friedrich Gauss when his elementary school teacher tried to keep his class busy by asking them to sum the numbers from 1 to 100, but was known at least as early as the 13^{th} century in Levi Ben Gershon's manuscript on *The Art of Calculation*.)

Problem 8: DFAs. Draw a DFA that recognizes the language of strings in $\{a,b\}^*$ that do not contain any occurrences of two consecutive *a*s. For example, ϵ, *abba* and *ababbbab* are in the language but *aa*, *baa*, and *bbabaab* are not. Use as few states as possible.

Challenge bonus: For a given maximum length n, how many strings of length less than or equal to n are in the language? (You are not expected to be able to solve challenge bonus problems, but will receive extra credit points for good solutions to them.)

Answer: See Class 5.

Problem 9: Lexical Analysis. DFAs are often used in compilers to recognize lexical units in programs. Tools such as flex (*http://flex.sourceforge.net/*) generate code that performs lexical analysis (breaking a stream of text into a sequence of tokens) by translating regular expressions into finite state machines.

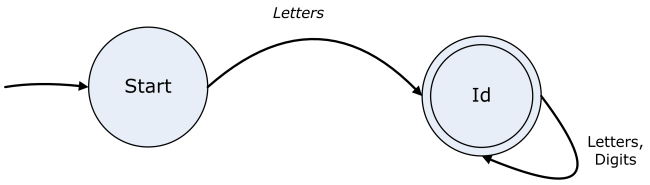
Many programming languages also provide library functions that perform scanning such as Java's `java.util.Scanner` class and Python's `re` library. Later in the course we will explore regular expressions and how a program might construct a DFA from a regular expression. In this problem, you will manually construct DFAs that recognize a few Java programming language constructs. For each part, your answer should be a description of a DFA that exactly recognizes the described language. You can either draw the complete DFA, or explain clearly how you would construct it.

For simplicity we assume an alphabet that is a subset of the actual Java alphabet:

$\Sigma = \mathit{Letters} \cup \mathit{Digits} \cup \{\text{Space}, \text{NewLine}, \texttt{/}, \texttt{*}\}$
$\mathit{Letters} = \{\texttt{a}, \texttt{b}, \texttt{c}, \ldots, \texttt{z}, \texttt{A}, \texttt{B}, \texttt{C}, \ldots, \texttt{Z}\}$
$\mathit{Digits} = \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}\}$

a. Identifiers: a sequence of one or more *Letters* and *Digits*, the first of which must be a letter.

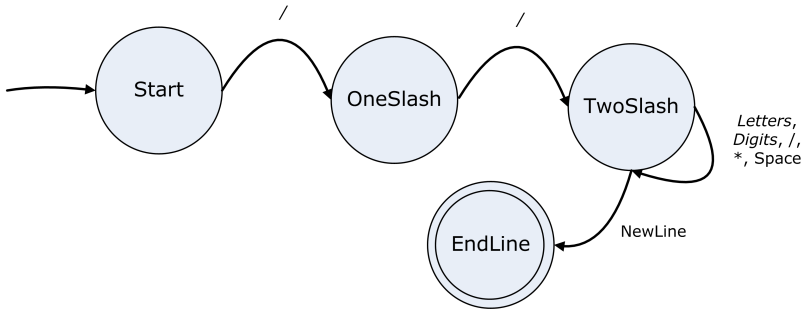
Answer: We need two states to keep track of whether or not we have already seen one *Letters*. We follow the convention here of not showing transitions that lead to rejection, so for if there is no transition in the current state on the input symbol, that means the machine transitions into a permanent reject state.



b. Single-line comments: a `//`, followed by any number of *Letters*, *Digits*, and Space characters until a NewLine.

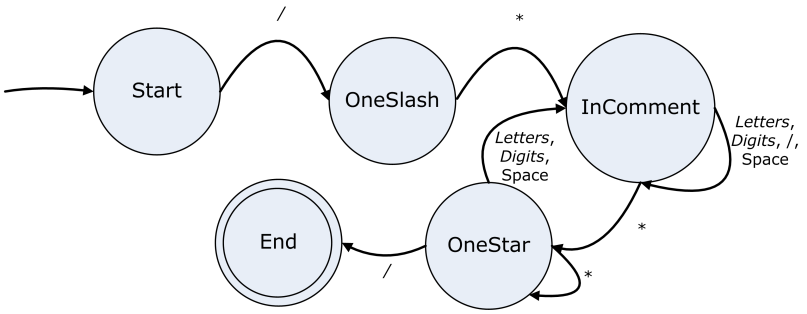
Answer: Lots of people had trouble with this one. Note that it is not a valid string in the language unless it ends with a NewLine, and that there are no

transitions out of the accepting state (hence, if there are any symbols after the NewLine, the input is not in the language).



c. Traditional comments: `/*` *comment* `*/` where *comment* is any element of Σ^* that does not contain the substring `*/`. (Note that comments do not nest, so `/* abcd /* // defg */` is a single comment.)

Answer: As with the previous sub-problem, a string is in the language only if it ends at the end of the comment. If we encounter a `*` within comment, we need a state to keep track of this, since the comment ends only if the `*` is followed by a `/`. We need a self-edge from the *OneStar* state on `*`, since `/* I like gold stars *******/` is a member of the language.



d. Any string in Σ^* that does not contain a comment (of either type).

Answer: The easiest way to answer this one is to explain how to construct the machine, rather than drawing it (which would involve quite a few states). The language is the complement of the union of the languages from the previous two parts. Since we have DFAs for the two parts, we could construct a DFA that recognizes the language of either comment by using the union construction (from the proof of Theory 1.25) on the two DFAs from the previous two subparts. Then, we can construct a DFA that recognizes the complement language, but setting the set of final states to the complement of F.