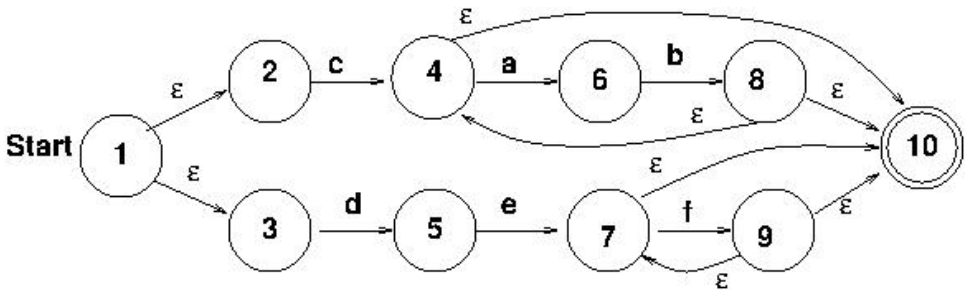
CSC173 FLAT
Module Exam

Chris Brown

Oct 2013

Please write your name on the bluebook. You may have two sides of hand-written notes. There are 75 possible points (one per minute). Best not to spend more minutes on a question than it's worth. Stay cool and please write neatly.

1 NDFA, Etc. (20 min.)



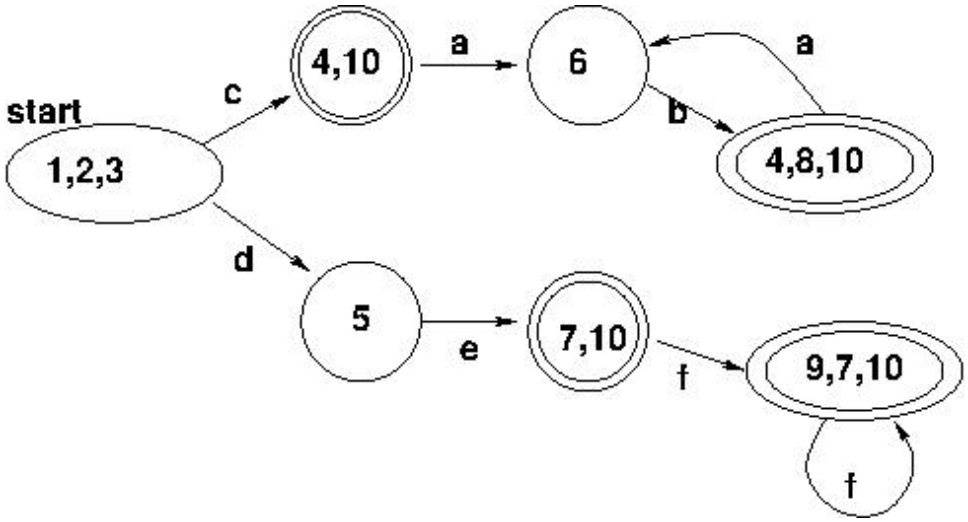
Above is an NDFA.

A. (5 min.) What is its corresponding regular expression?
B. (10 min.) Use the subset construction to convert the NDFA to a DFA.
C. (5 min.) Is the resulting DFA minimal (fewest states possible)? Argue one way or the other.

Ans.

A. $c(ab)* \mid def*$

B.



FFQ: It's too easy to just use your head and make up an optimal 5-state machine and write that down here. That does not illustrate you've learned the subset construction. As ever: process, not product.

However, one can certainly use such a made-up machine to illustrate that the subset-construction DFA is not minimal. That's a fine way to avoid confronting that minimization algorithm (below).

C. Not minimal; Upper and lower-branch terminal states can be merged, yielding a 5-state solution with two terminal states. In the original diagram, just replace the 7,10 state with the 9,7,10 state, delete the 4,8,10 state and instead add a "b" transition from 6 to terminal 4,10. Merging the two terminal states in this solution would allow, for instance, cababababfffabfabff, which isn't c(ab)* or def*.

2 Parsing (25 min.)

Here's a CFG with starting symbol G, non-terminals G, S, A, B, C, D; terminals a, b, c, d; ϵ means the production produces the null string. $< eof >$ is a single end-of-file token.

$G \rightarrow S < eof >$
$S \rightarrow SAB$
$S \rightarrow AC$
$S \rightarrow AD$
$A \rightarrow a$
$B \rightarrow b|\epsilon$
$C \rightarrow c|\epsilon$
$D \rightarrow d|\epsilon$

A. (10 min.) Is this an LL (left to right, left derivation) grammar? If not please say why not and fix it so it is LL.

B. (5 min.) For the original grammar, what is the FIRST set of S? What is the FOLLOW set of A?

C. (5 min.) Is the original grammar ambiguous? Show why or why not. You can forget G and concentrate on what's derivable from S.
D. (5 min.) Show a parse tree from the original grammar for the string "$acab$". You can forget G and concentrate on S.

Ans:

A. No it's not LL because of the left recursion on S and common prefix with A. Fixing the left recursion is a bit tricky – we have to interpret the re-writing rule for left-recursion elimination as if
$S \rightarrow SAB|AC|AD$
is really $S \rightarrow SAB|(AC|AD)$.

With that, we remove the left recursion thus:
$S \rightarrow SAB|AC|AD$
$\Rightarrow$
(1) $S \rightarrow ACX$
(2) $S \rightarrow ADX$
(3) $X \rightarrow AB|\epsilon$

Now we have the problem of S having two productions with a common prefix. Applying the rule for that we get:
(1) $S \rightarrow ACX$
(2) $S \rightarrow ADX$
$\Rightarrow$
$S \rightarrow AY$
$Y \rightarrow CX$
$Y \rightarrow DX$
(3) $X \rightarrow AB|\epsilon$

So the LL grammar is:
$G \rightarrow S < eof >$
$S \rightarrow AY$
$Y \rightarrow CX$
$Y \rightarrow DX$
$X \rightarrow AB|\epsilon$
$A \rightarrow a$
$B \rightarrow b|\epsilon$
$C \rightarrow c|\epsilon$
$D \rightarrow d|\epsilon$

I rather think that removing the common prefix first is a more obvious and straightforward approach leading to a slightly different grammar. Exercise for the reader to do that.

B. FIRST(S) = {a}, FOLLOW(A) = {a,b,c,d, $< eof >$}. However, there is no $A \rightarrow |\epsilon$ production so I'm not sure what the rule is. FOLLOW(A) = {b,c,d, $< eof >$} (or even FOLLOW(A) = {b,c,d}, which is actually practically wrong, were full-credit answers.

FFQ: The question is FIRST(S), FOLLOW(A) – not FIRST and FOLLOW of S.

C. The typo that repeated the B production didn't make grammar ambiguous: redundant and repetitious, yes. It's the same B in the same parse tree with either production.

One can deduce ambiguity by looking at the predict sets and assuring no pairwise non-null intersections, but on grammars this size it's probably easier just to find an example.

It's ambiguous: Leaving G and its $< eof >$ out of it, $S \rightarrow AC \rightarrow a\epsilon$ and $S \rightarrow AD \rightarrow a\epsilon$ both yield a. (Same's true of the transformed grammar, as it should be, with slightly longer derivation: $S \rightarrow AY \rightarrow aCX \rightarrow a$ (with C and X both taking the epsilon option) and $S \rightarrow AY \rightarrow aDX \rightarrow a$ (with D and X both taking the epsilon option.)

D. Forgetting G,
$S \rightarrow SAB \rightarrow ACAB \rightarrow acab$. There's (there'd better be!) a (bigger) parse tree for this string using the LL grammar you made. Could be a good check on the work.

3 FLAT Potpourri (30 min.)

A. (5 min) Show that the set of palindromes (strings same as their reverses) over $\{0, 1\}$ is not regular (can't be accepted by a DFA, has no regular expression).

B. (5 min) A deterministic finite-state automaton has state set S. Show that the language it accepts has an infinite number of words (strings of symbols) if and only if there is a word x recognized by the machine whose length is $\geq |\ S\ |$, where $|\ S\ |$ is the size of (number of elements in) the set S.

C. (10 min.) As usual, Σ is a set of symbols, $L \subseteq \Sigma^*$ a language (set of strings of symbols). Define L^x as $\{z \in \Sigma^* \mid xz \in L\}$: z is the set of strings that if appended to x make a sentence of L. We say $x, y \in \Sigma^*$ are *distinguishable with respect to L* if $L^x \neq L^y$. Thus a string z *distinguishes* x and y if $xz \in L$ but $yz \notin L$, or *vice-versa*. If $L^x = L^y$, x and y are *indistinguishable* with respect to L.

Let L be the set of all strings over $\{0, 1\}$ ending in 01.
C.1: Show that (find a z such that) 11 and 10 are distinguishable with respect to L.
C.2: Show that the strings 1 and 11 are indistinguishable with respect to L.

D. (10 min.) Extend the definition of a DFA's transition function from $T(S_1, a) \rightarrow S_2$, telling us what state results from being in S_1 and reading a, to $T(S_1, x) \rightarrow S_2$, telling us what state results from being in S_1 and reading a string of symbols x. So now T tells us what state we transition to, given a starting state and a string of input.

For a DFA M with starting state S_0 and accepting the language $L(M)$, show that if x and y are two strings in Σ^* that are distinguishable with respect to $L(M)$, then $T(S_0, x) \neq T(S_0, y)$.

Ans: (These are all from Rosen, btw...)

A. An n-state DFA can only "remember" (have a separate state for) an (n-1)-long input string. For palindrome detection, we must remember strings of at least m/2 for an m-long palindrome, and m can be arbitrarily large, so can overwhelm any DFA of any fixed n.

B. If an n-state DFA accepts an infinite number of words, one of them must be longer than n. An n-state DFA can have only n-1 different transitions before it has to repeat a state (pigeonhole principle). And if it repeats a state it's in a loop, with nothing to stop it running forever. So accepting a word of length $\geq n$ implies accepting an infinite number of words.

C.1. If z = 1, then 111 is not in L, 101 is, so they're distinguishable.
C.2. To put either 11 or 1 into L, we have to add a z that ends in 01. But doing that puts them both in, so we can't make a sentence in L that starts with 1 that's in the language by appending something that doesn't also put 11 in the language; they're indistinguishable.

D. If x and y both put us in the same state, then the rest of the input (i.e. z) determines whether the whole input string is accepted. Thus they're either both in or both out. Thus they're indistinguishable.

FFQ: C and D: "Show me your z's!"