# Formal Languages and Automata Theory

Chris Brown

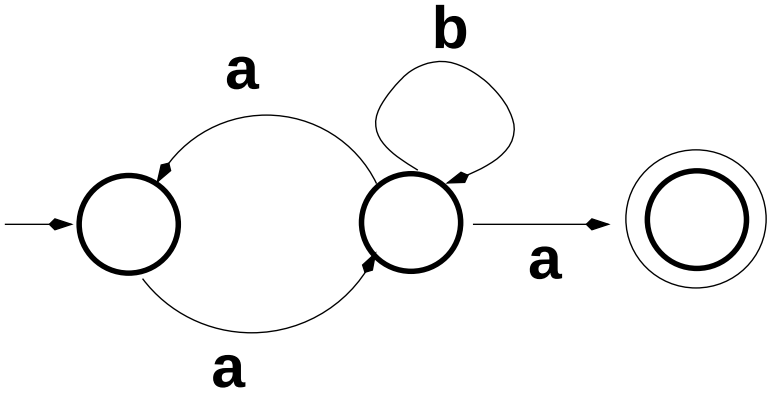
## 1 Week 2:

*Do NOT attempt to submit these assignments through Blackboard: you can't. Use the turn-in box outside Marty's office.*

2.2.1. Design an automaton to read strings from the [0,1] alphabet, and to accept any string that has no more than two consecutive 1's. That is, accept unless 111 is a substring of the input string read so far (or, reject as soon as, but only when, you see 111 in the input). The zero-length (null) string is therefore legal.

2.2.2.Design an automaton that tells whether a given character string is one of the third-person singular pronouns `he, his, him, she, her, hers`, followed by a blank.

2.2.3. Describe in English the language of the following NDFA.




2.2.4. Write the regular expression for the previous NDFA.

2.2.5. Using subset construction, convert the NDFA to a DFA.

2.2.6. Describe in English the language of the following regular expression.

`(ab)* (ba)* | ab*`

2.2.7. Convert the previous regular expression into an NDFA with epsilon transitions.

2.2.8. Draw a deterministic finite automaton with four states that recognizes the strings with the alphabet {`a,b`} in which the number of times `ab` appears is even.

# 2 Weeks 3 and 4:

2.4.1. Write a CFG for all strings containing 0 and 1 that contain the same number of 0s and 1s. Show parse trees for the strings 001101 and 0110.

2.4.2. Show that the following grammar (where the last production is an epsilon production and lower-case symbols are terminals) is ambiguous:

$S \rightarrow$ `a` $S$ `b` $S$
$S \rightarrow$ `b` $S$ `a` $S$
$S \rightarrow \epsilon$

2.4.3. Informally, $First(A)$ is defined to be the set of tokens that can begin some string of tokens derived from $A$. Compute $First(Stmt)$ for the following grammar fragment:

$Stmt \rightarrow LabeledOpt\ Loop$
$Stmt \rightarrow$ `indent :=` $Expr$
$LabeledOpt \rightarrow$ `number :`
$LabeledOpt \rightarrow \epsilon$
$Loop \rightarrow$ `while` $Condition$ { $StmtList$ }
$Loop \rightarrow$ `do` { $StmtList$ } `until` $Condition$

2.4.4. Remove left recursion from the following grammar:

$IdList \rightarrow IdListPrefix$ ;
$IdListPrefix \rightarrow IdListPrefix$ , `id`
$IdListPrefix \rightarrow$ `id`

2.4.5. Perform *left factoring* on (that is, remove common prefixes from) the following:

$Stmt \rightarrow$ `id :=` $Expr$
$Stmt \rightarrow$ `id (` $ArgumentList$ `)`

Why do you think this operation is called left factoring?

2.4.6. Formally prove that the following grammar is LL(1) using first and follow sets:

$S \rightarrow$ ( $L$ )
$S \rightarrow a$
$L \rightarrow SLtail$
$Ltail \rightarrow$ , $L$
$Ltail \rightarrow \epsilon$