Programming Language Syntax

Chapter 2, Translating Regular Expressions to a DFA

Objective

Today's primary objective:

- Define the terms nondeterministic finite automaton (NFA) and deterministic finite automaton (DFA)
- Know how to go from a regular expression to an NFA and then to a DFA

Create Scanning Table

Steps to create a table for the lexical analyzer

1. Create regular expressions for each token
2. Create a Non-deterministic Finite Automaton (NFA) for all
3. Convert NFA to a Deterministic Finite Automaton (DFA)
4. Convert the DFA to a minimal DFA
5. Build table for lexical analyzer code.

Finite Automaton

FA - “machine” representing a regular expression

- Used to show acceptance and non-acceptance of strings (depending on whether the string matches the regular expression or not)
- Consists of states and arrows between states (transitions)
- One start state (arrow from nothing going into it)
- Any number of accept states

DFA versus NFA

Deterministic Finite Automaton, DFA

- For each state there is exactly one transition leaving it for each symbol in the alphabet

Non-deterministic Finite Automaton , NFA

- For each state there may be many or no transition leaving it. Also ε -transitions are allowed (transitions that can be taken without consuming a symbol)

Step 1.) Create regular expressions for each token

This was covered in the previous section.

Step 2.) Create an NFA for all

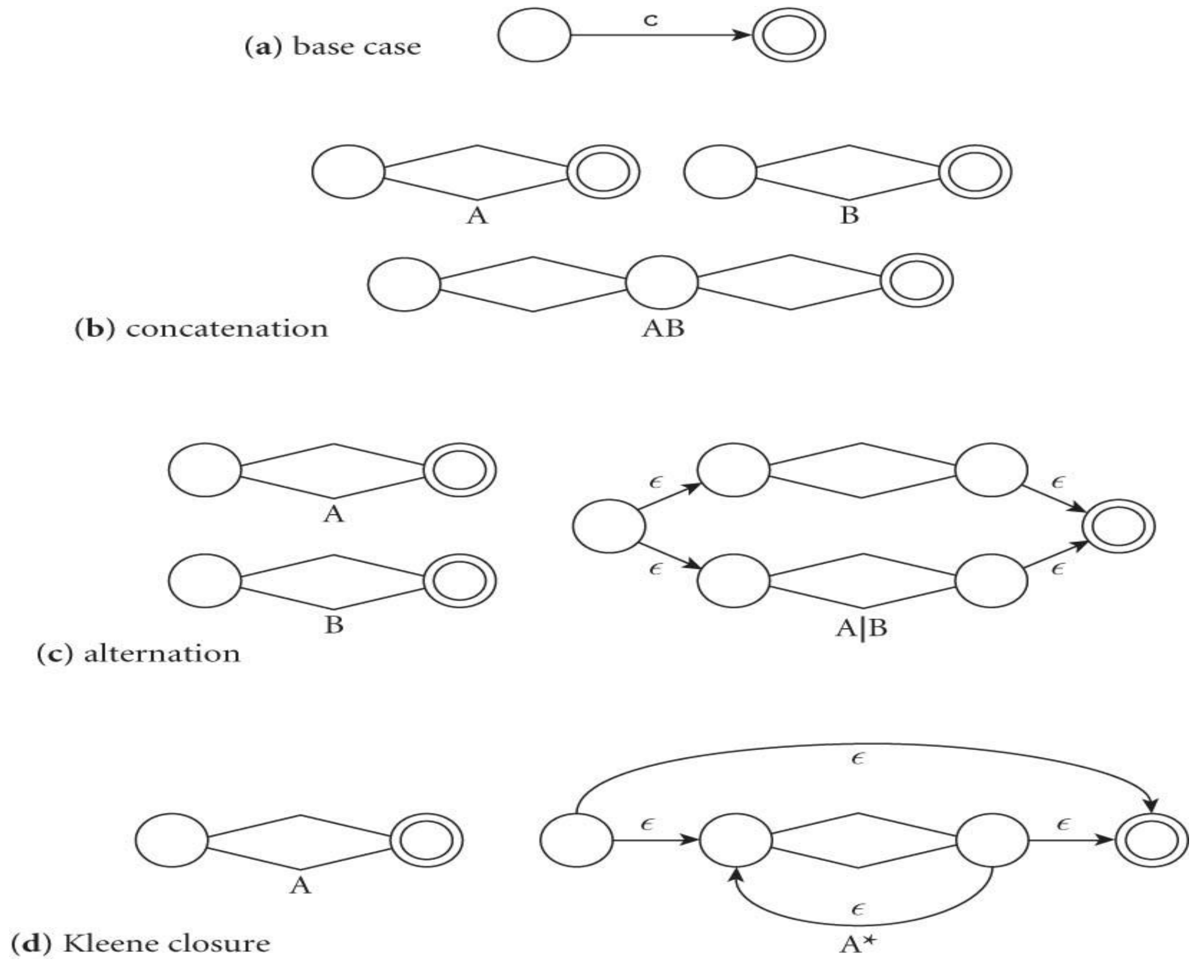


Figure 2.7, page 56 from te

Step 3.) Convert NFA to a DFA

Algorithm to go from an NFA to a DFA:

In general make the states of the DFA represent the set of states that the NFA could be possibly be in.

1. Create an initial state which consists of the initial state of the NFA along with all of the states that the NFA could have been in initially (i.e. follow epsilon transitions if possible)
2. Repeat until transitions have been defined for every state in the DFA on every input.

   For each state q (represented by a set of states) and each input a, create a transition from q on a which goes to a state representing all states that the NFA could have been in if it started in a state in q and saw the input a. Be sure to follow epsilon transitions as well.
3. Mark any state whose representation contains a final state in the NFA as final.

Page 56 & 57 from text

Step 4.) Convert the DFA to a Minimal DFA

Algorithm for converting a DFA to a minimal DFA:

1. Place the states of the DFA into two equivalence classes: final states and non-final states.
2. Repeat until no more classes to partition:

    Search for an equivalence class X and an input symbol c such that when given c as input, the states in X make transitions to states in k>1 different classes. Partition X into k classes so that all states in a given new class move to a member of the same class on c.

Page 59 from text

Example – Simple Calculator Language

Regular Expressions for the tokens

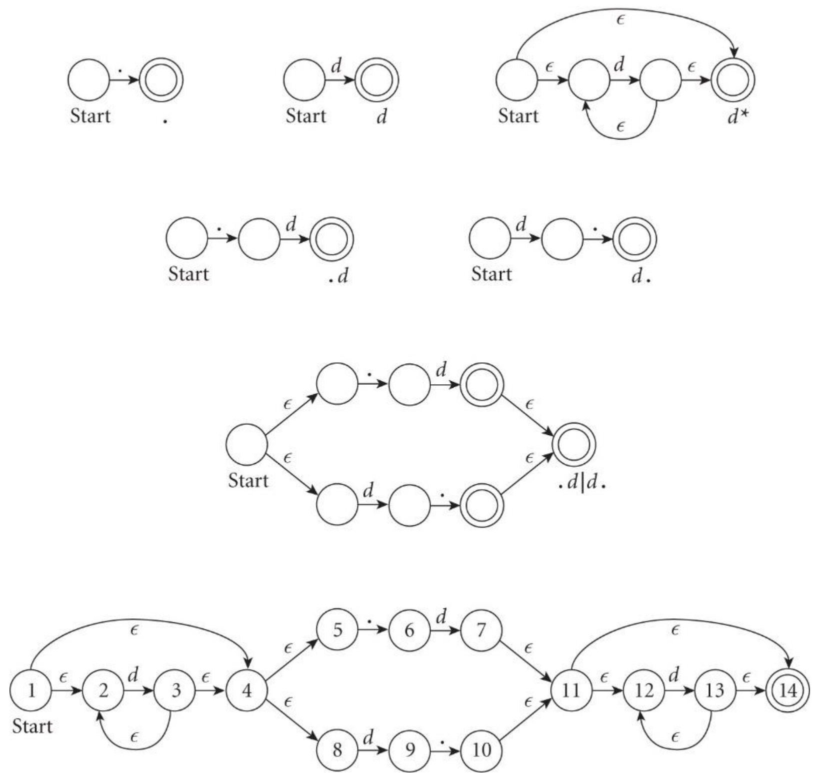
assign → :=
plus → +
minus → -
times → *
div → /
lparen → (
rparen →)
id → letter (letter | digit)*
number → digit digit* | digit* (. digit | digit .) digit*

Floating Point Number

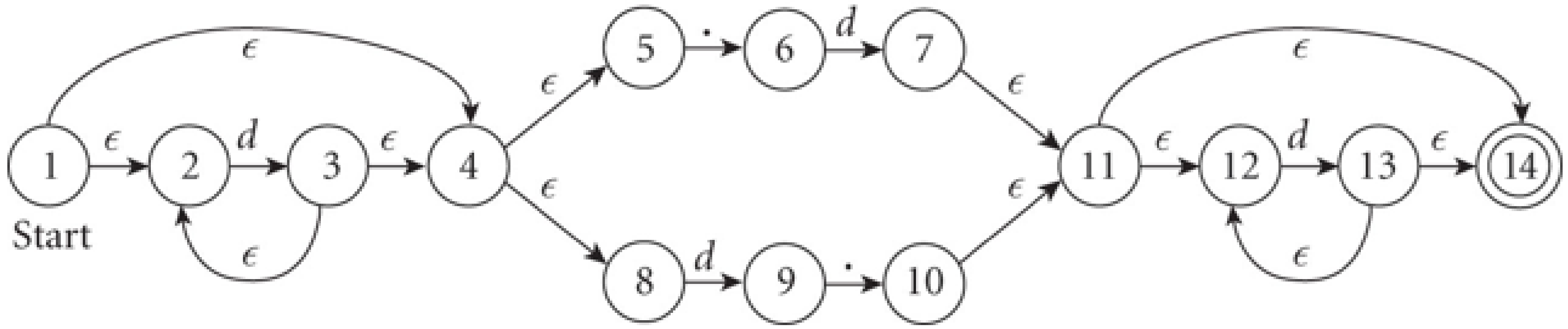
number → digit* (. digit | digit .) digit*

Regular Expression to NFA

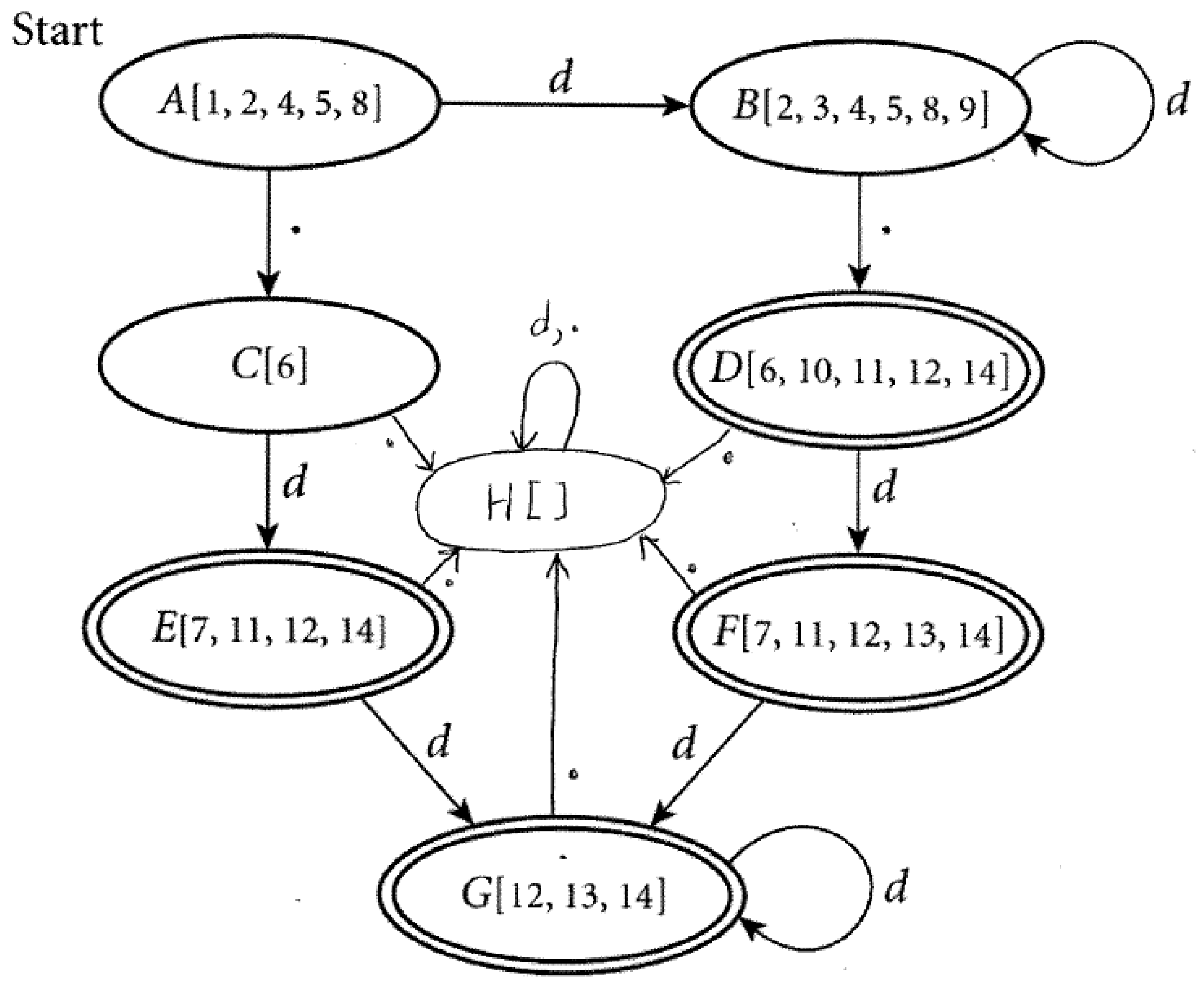
number → digit* (. digit | digit .) digit*



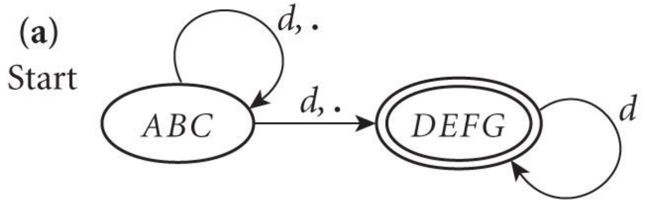
NFA

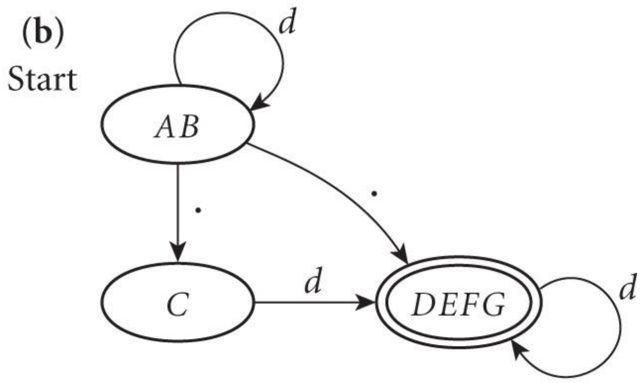


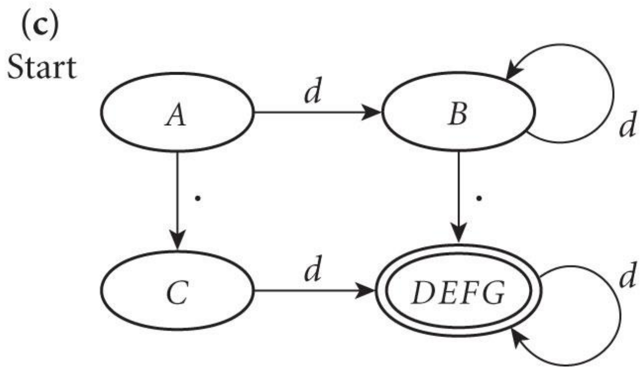
Equivalent DFA



To Minimal DFA


(a)
Start
ABC
d, .
d, .
DEFG
d


(b)
Start
AB
d
.
.
C
d
DEFG
d


(c)
Start
A
d
B
d
.
.
C
d
DEFG
d