Note Title 1/24/2012

CS 344 - Scanning

Announcements

- Essay due on Friday
- Next HW will be up by Friday

$d^+ = d(d)^*$

Last time: Regular expressions

- A character
- The empty string, ε
- 2 regular expressions concatenated
- 2 regular expressions separated by an or (written $|$)
- A regular expression followed by $*$ (Kleene star - 0 or more occurrances)

$+$ operator

$(d)^+$

1 or more occurances

Ex: Give the regular expression for
$\{ w \mid w$ begins with a 1 and ends with a 0 $\}$

$$1(0|1)^*0$$

Ex: $\{ w \mid w$ starts with 0 and has an odd length $\}$

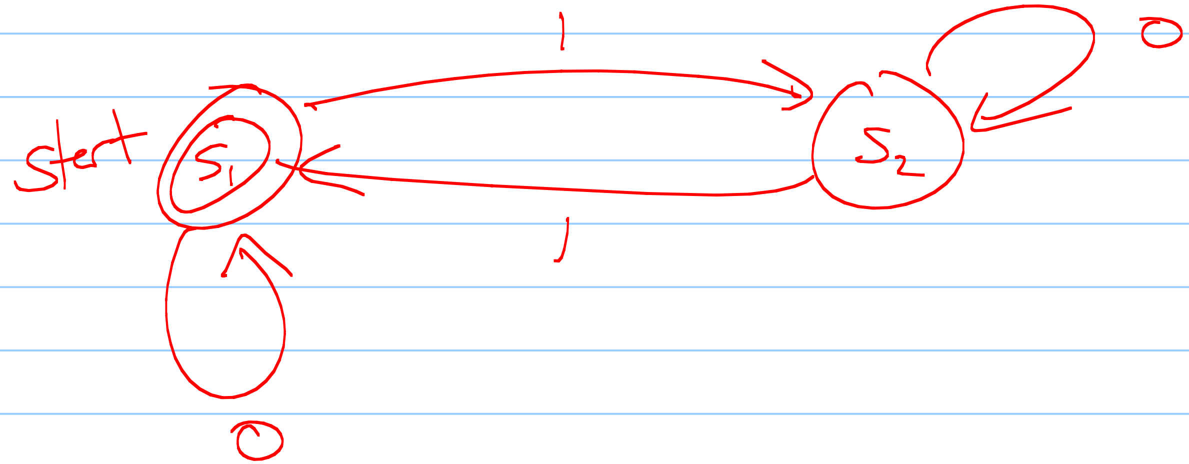
$$0((0|1)(0|1))^*$$

Deterministic Finite Automata (DFA)

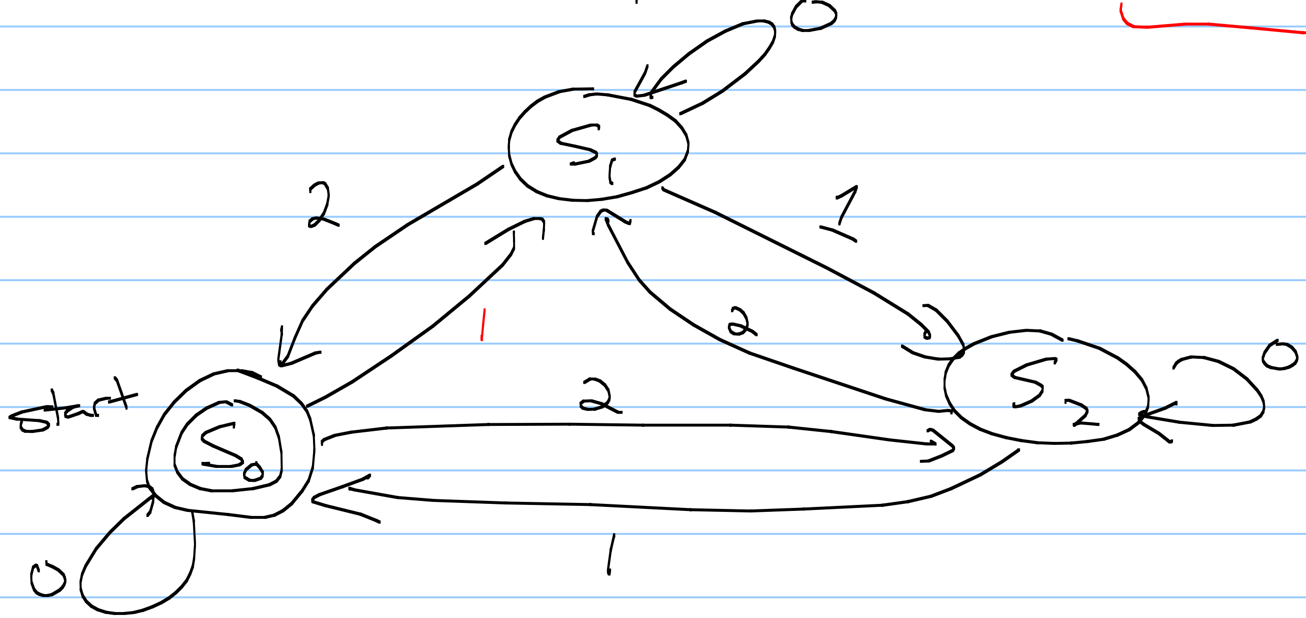
Regular languages are precisely the things recognized by DFAs.

- A set of states
- input alphabet
- A start state
- A set of accept states
- A transition function: given a state & an input, output a new state

Ex: String of 0's + 1's:
accept if number of 1's is
even



Ex: 3 symbol alphabet: $\{0, 1, 2\}$

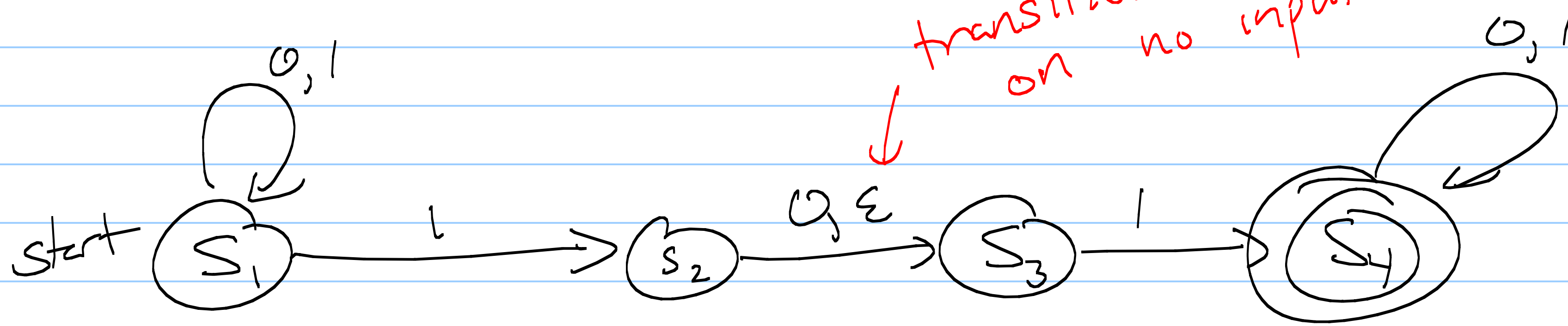


computing mod 3

accepting if sum of word is 0 mod 3

NFAs: DFAs w/ ambiguity

transitions on no input

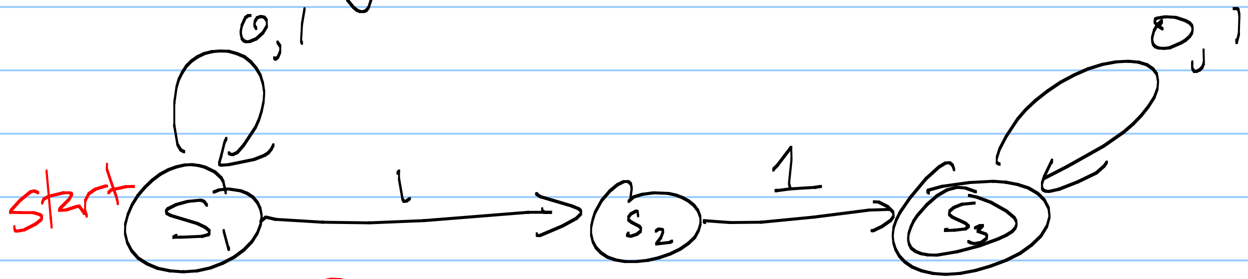


if read in a 1 could go to S_1 or S_2

NFA has n states
DFA has up to 2^n states

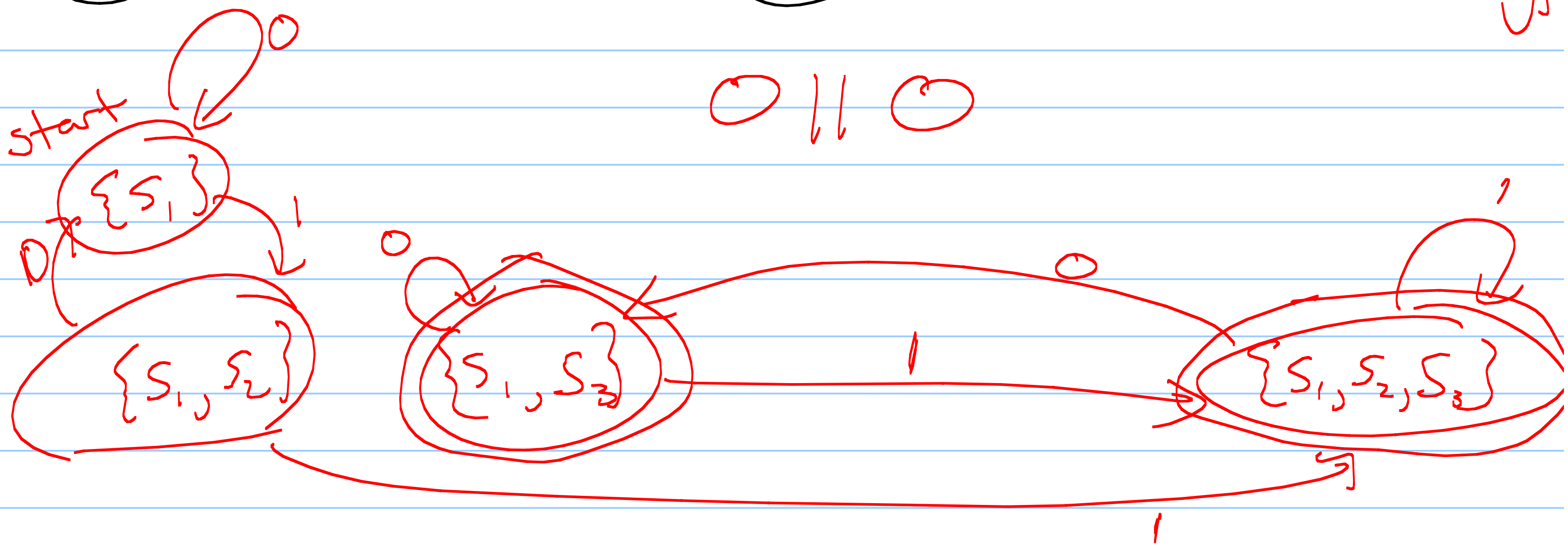
Converting NFAs to DFAs

$\{S_1, S_2, S_3\}$



$\{w \mid w$ contains 11 as a substring$\}$

0110



Context free Grammars (~BNF)

Ex:

expr → expr op expr | (expr) | − expr |
id | number

terminals → id variable
number int/float

x = 5
y = 2

op → + | − | * | /

terminals

variables (ids)

<u>A derivation</u>: derive slope * x + intercept

expr $\Rightarrow$ expr op expr

$\Rightarrow$ expr + expr

$\Rightarrow$ expr + id (intercept)

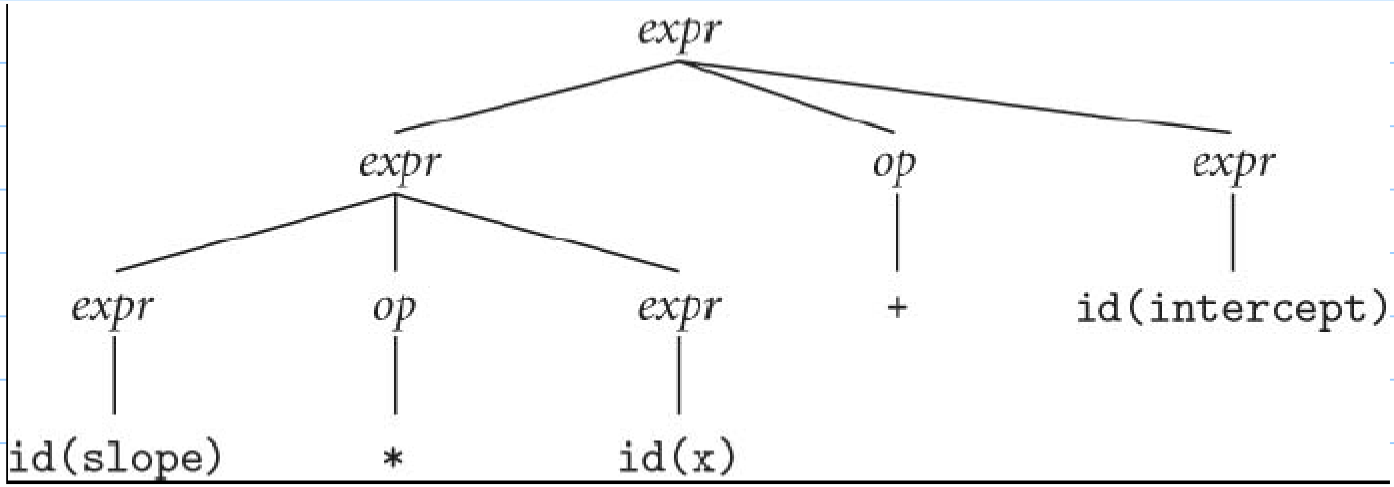
$\Rightarrow$ expr op expr + id

$\Rightarrow$ expr * expr + id

$\Rightarrow$ expr * id (x) + id

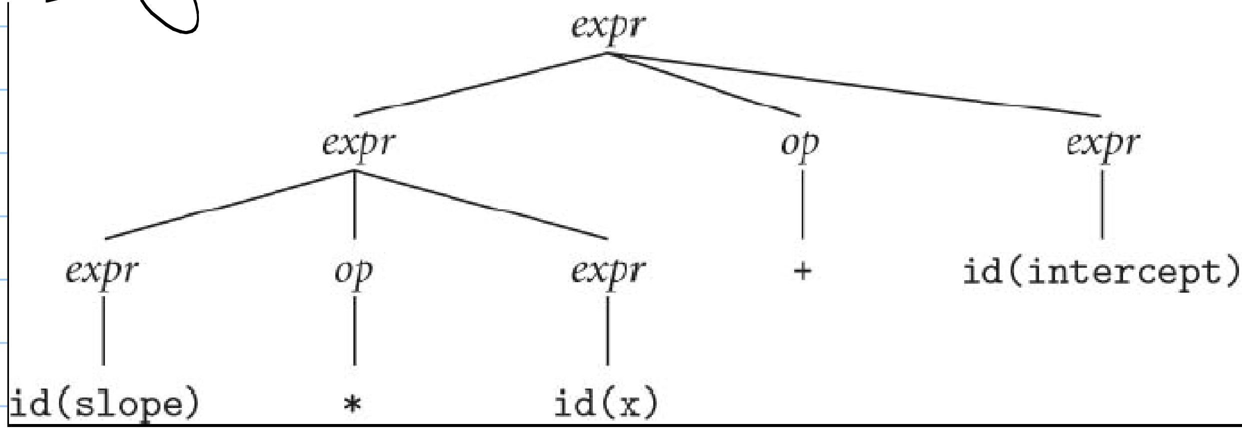
$\Rightarrow$ id (slope) * id + id

Derivation tree

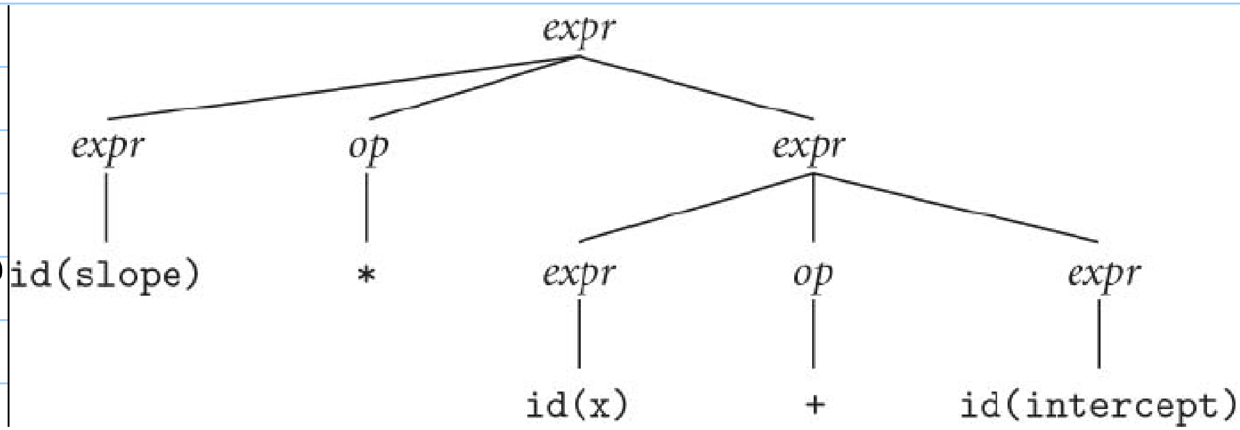


(rightmost derivation)

Ambiguous grammars



leftmost derivation



Grammars

There are infinitely many ways to make a grammar for any Context free language.

Problem in the pasing stage: which is better?

(Try to define unambignous grammars.)

Another example (from last time)

Expression grammars: Simple calculator

expr → term | expr add_op term

term → factor | term mult_op factor

factor → id | number | - factor | (expr)

variable

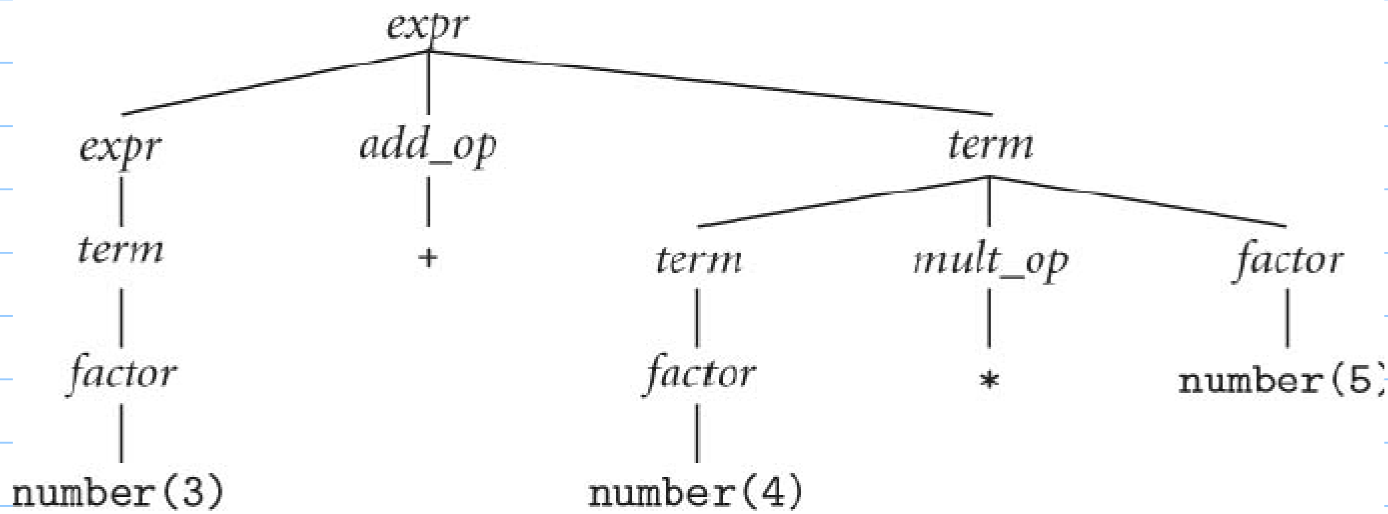
add_op → + | -

terminals

mult_op → * | /

Parse Tree

Ex: 3 + 4 * 5



Scanners

Find the syntax (<u>not</u> semantics) of code.

Output tokens.

<u>3 types</u>

- Ad-hoc
- Finite automata
  - ~ nested case statements
  - ~ table + driver

Ad-hoc (last time)

if current ∈ { "(", ")", "+", "−", "*" }
    return that symbol
if current = ":"
    read next
    if it is =, announce "assign"
        else announce error
if current = "/"
    read next
    if it is "*" or "/"
        read until "*/" or "newline" (resp.)
    else return divide
etc.

Ad-hoc approach

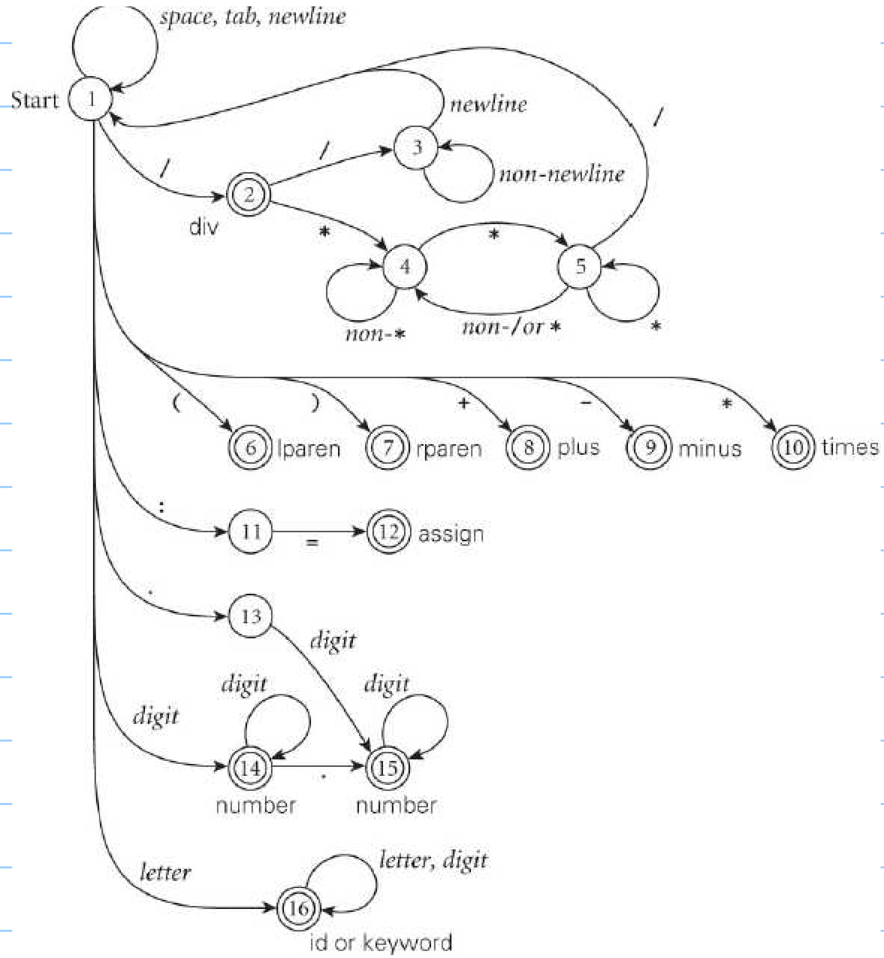
Advantage:
code is fast & compact

Disadvantage:
very ad-hoc!
- hard to debug
- no explicit representation

DFA approach

Recall our simple calculator language.

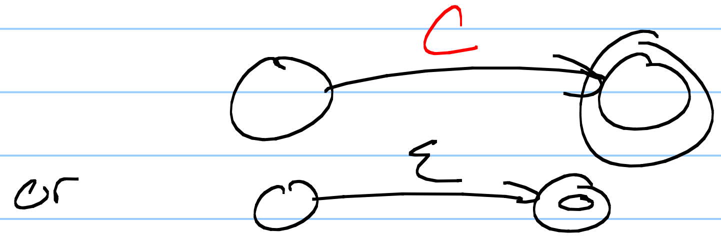
But how to get this DFA & then how to actually model it?



Constructing a DFA

Given a regular expression, we can construct an NFA.

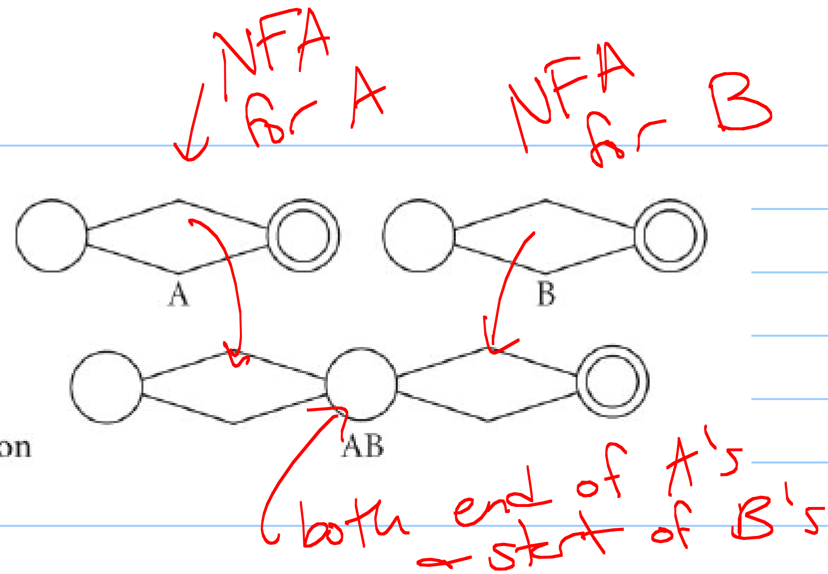
Simple NFA:



(Base case)

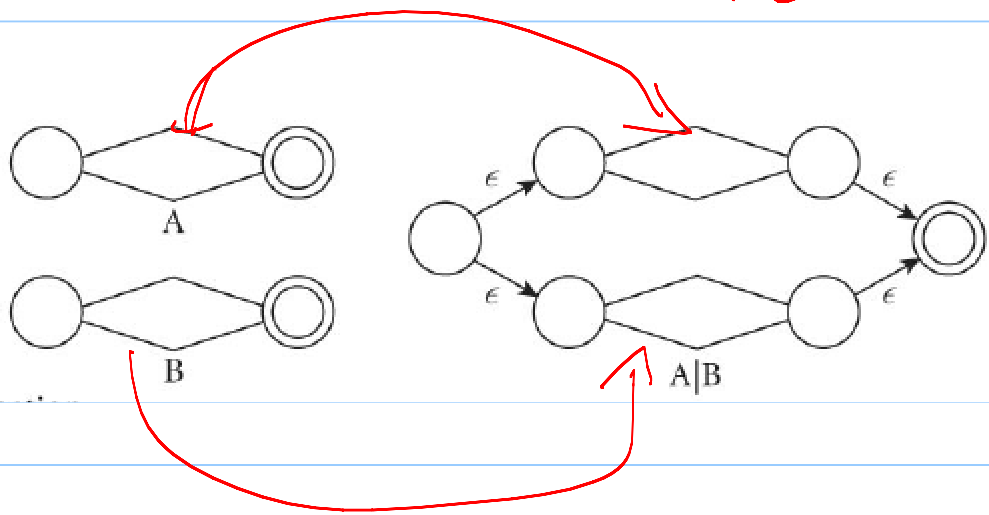
3 operations

Concatenation:

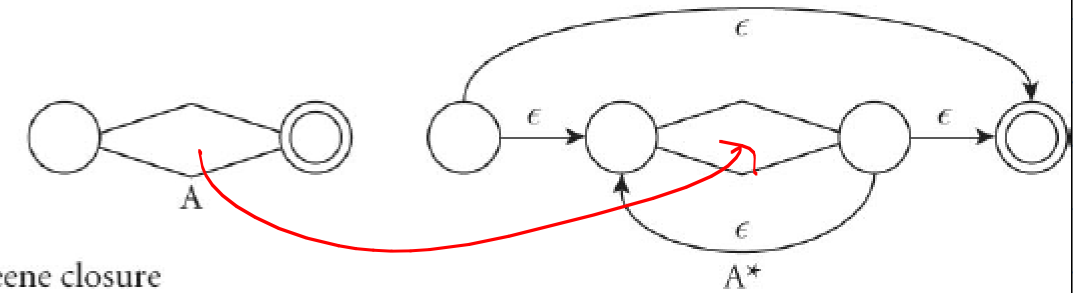


concatenation

Or:

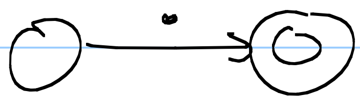


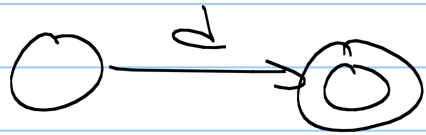
and Kleene closure (*).

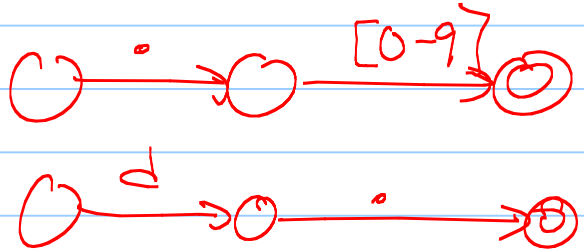


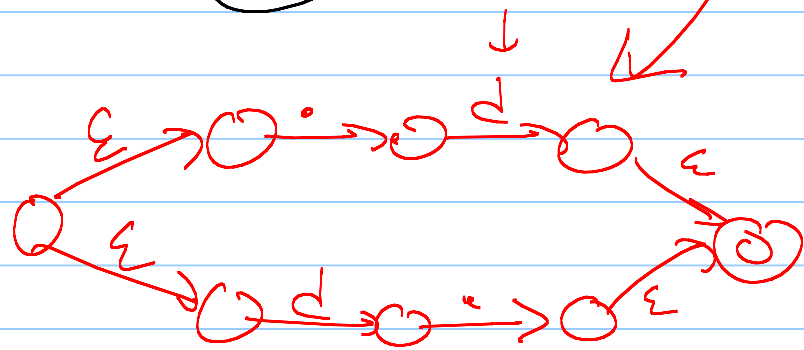
(d) Kleene closure

Example: decimals $d^*(.d \mid d.)d^*$

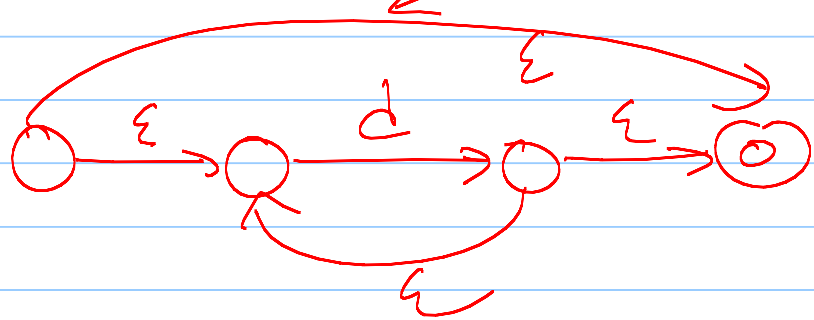
Base: 



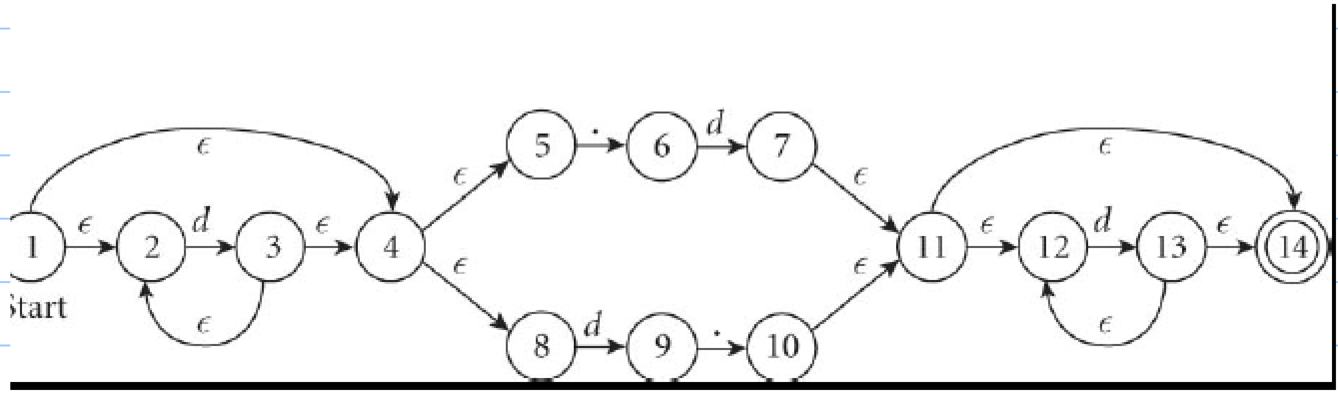




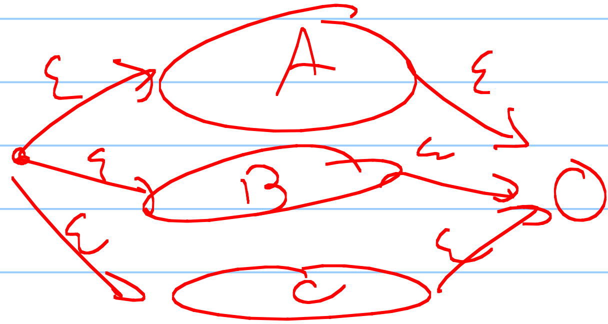
d^*:



Final product:



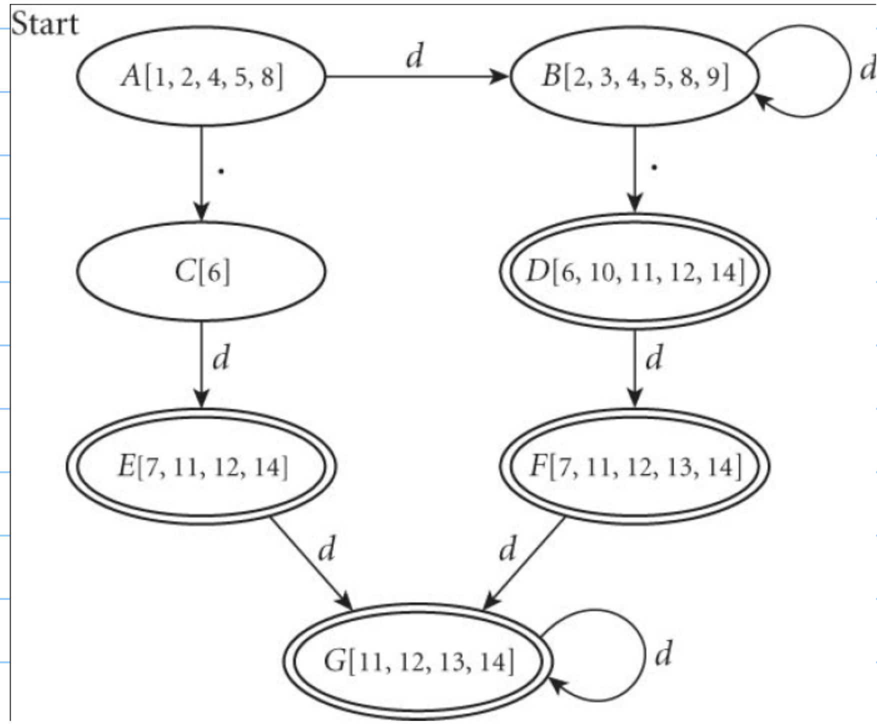
A | B | C



Next: Convert to DFA.
(Lots of states, but same principle as we saw earlier.)

Result:

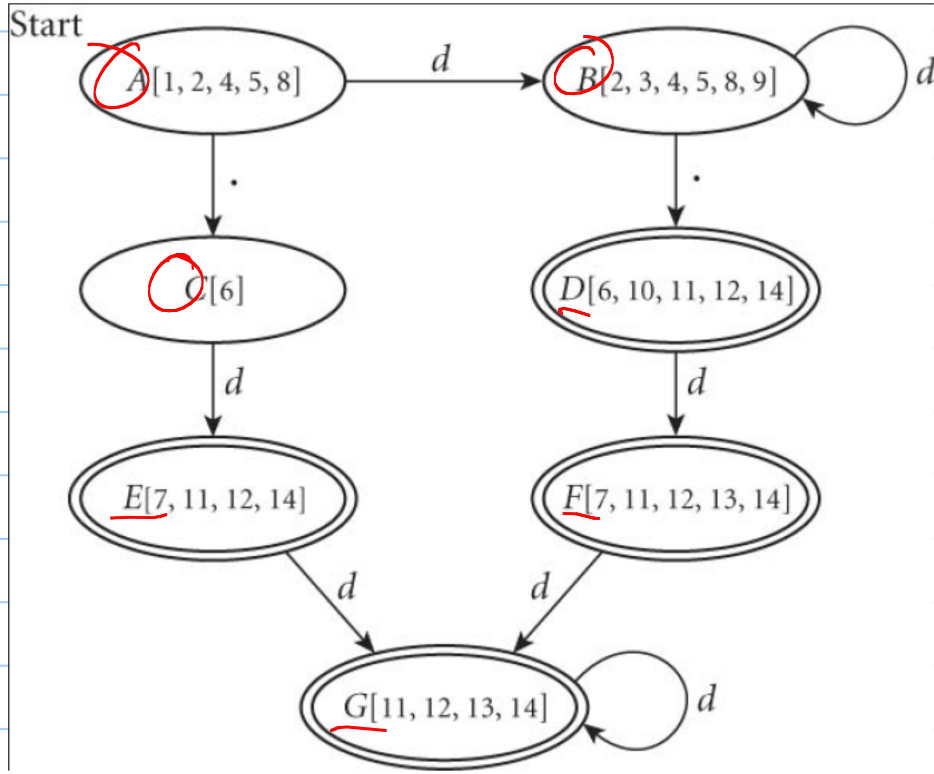
(see p. 57-58)



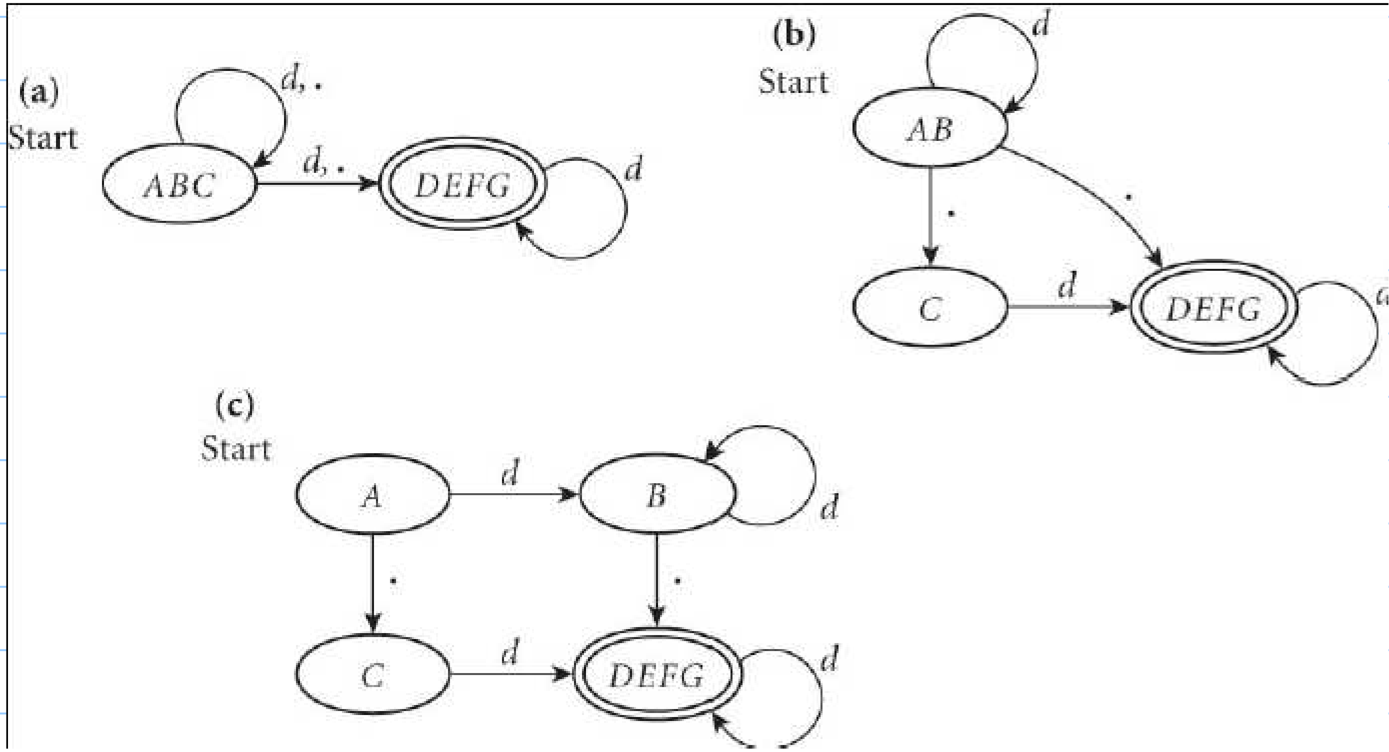
Note: This DFA is a bit redundant.

Not minimal.

Can easily find the equivalence classes and minimize.



Process to minimize



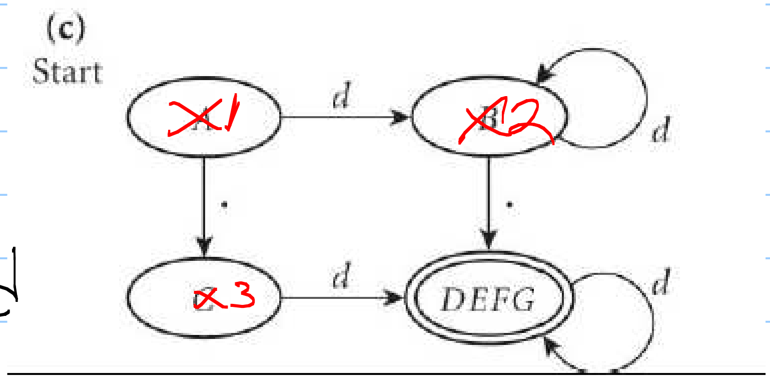
Now:

Given DFA, generate case statements to simulate it.

state = 1
repeat;
  read curr_char
  case state is:

    1: case curr_char = d
         state = 2
       case curr_char = .
         state = 3

    2:



Scanner Tools

In reality, this DFA is often done automatically.

Specify the rules of regular language, & the program does this for you.

Many such examples:

Lex (flex), Jlex / Jflex, Quex, Ragel, ...

Next time:

Lex / flex : C-style driver

Look for HW on regular expressions, NFA / DFA, & context free languages

Next programming assignment will use flex.