NFAs to DFAs

Today

- NFAs to DFAs

Logistics

- HW1 has been posted and is due Monday.
- Plan to put up videos today.

(Non) Deterministic Finite State Automata

A Deterministic Finite State Automaton (DFA) has disjoint character sets on its edges, i.e. the choice "which state is next" is deterministic.

A Non-deterministic Finite State Automaton (NFA) does NOT, i.e. it can have character sets on its edges that overlap (non empty intersection), and empty sets on the some edges (labeled ε).

NFAs are used in the translation from regular expressions to FSAs. E.g. when we combine the reg. exp for IF with the reg.exp for ID by just merging the two Transition graphs, we would get an NFA.

NFAs are a first step in creating a DFA for a scanner.

The NFA is then transformed into a DFA.

From regular expressions to NFAs

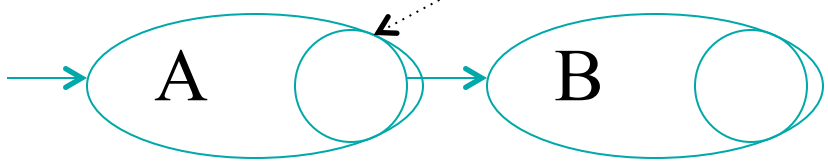
regexp

simple letter "a"

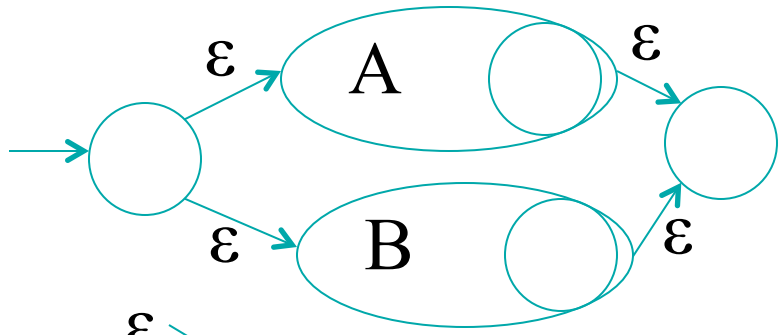
empty string

accept state of the NFA for A

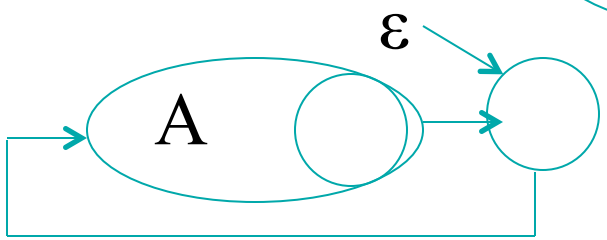
AB concat the NFAs



A|B split merge them



A* build a loop



The Problem

DFAs are easy to execute (table driven interpretation)

NFAs are easy to build from reg. exps, but hard to execute

we would need some form of guessing, implemented by back tracking

To build a DFA from an NFA we avoid the back track by taking all choices in the NFA at once, a move with a character or ε gets us to a set of states in the NFA, which will become one state in the DFA.

We keep doing this until we have exhausted all possibilities.

This mechanism is called transitive closure

(This ends because there is only a finite set of subsets of NFA states. How many are there?)

Example IF and ID

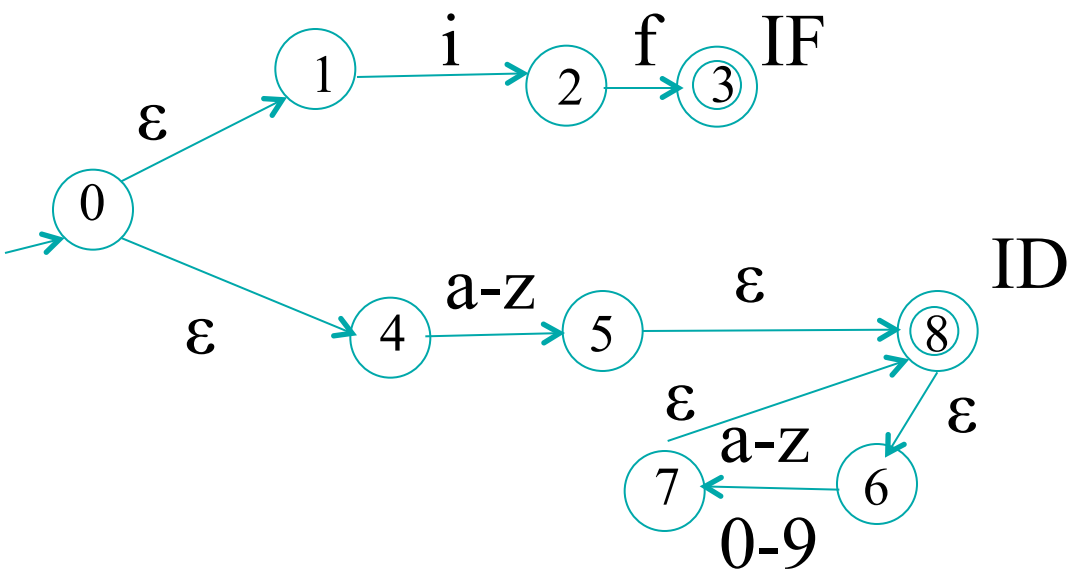
let : [a-z]
dig : [0-9]

tok : if | id

if : “i” “f”

id : let (let | dig)*

Example: NFA for IF and ID



let : [a-z]

dig : [0-9]

tok : if | id

if : "i" "f"

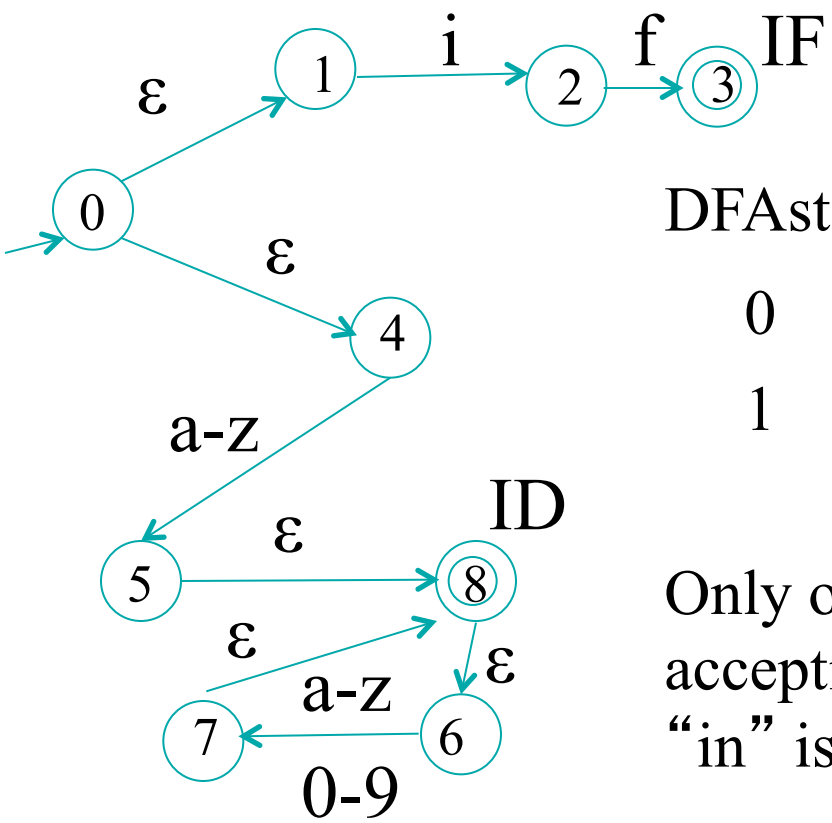
id : let (let | dig)*

From 0, with ε we can get to states 1 and 4

this is called an **ε-closure**

We can now simulate the behavior of the NFA and build a table for the DFA making character moves plus ε-closures

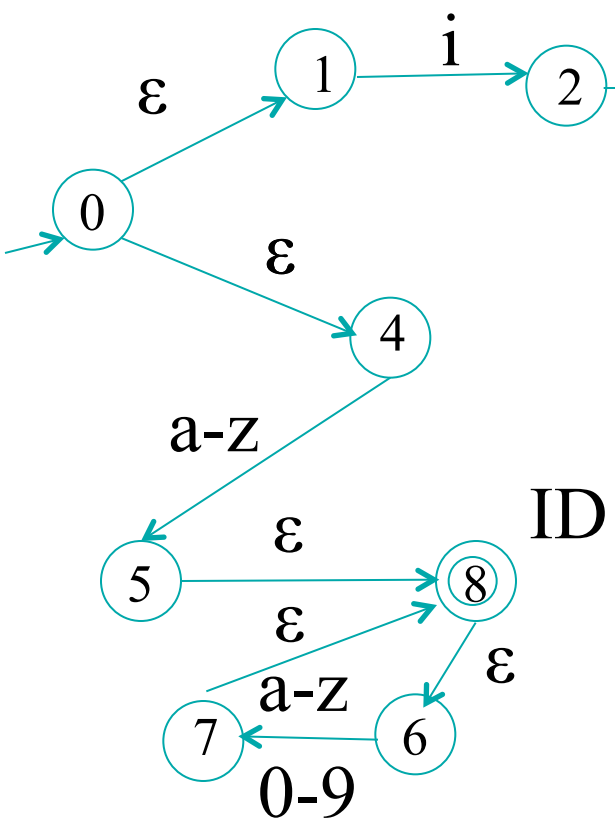
NFA simulation scanning "in"



DFAstate	NFAstates	Move	Next
0	0,1,4	i	2,5,8,6
1	2,5,6,8	n	6,7,8

Only one of the states in 6,7,8 is an accepting state, an ID accepting state, so "in" is an ID

NFA simulation scanning "if"



DFAstate	NFAstates	Move	Next
0	0,1,4	i	2,5,6,8
1	2,5,6,8	f	3,6,7,8

Two of the states in 3,6,7,8 are accepting, an IF accepting state (3) and an ID accepting state (8), IF has priority over ID, so "if" is an IF

Definitions: edge(s,c) and closure

edge(s,c): the set of all NFA states reachable from state s following an edge with character c

closure(S): the set of all states reachable from S with no chars or ε

T=S

repeat T' =T;

$$T = T' \cup \left(\bigcup_{s \in T'} edge(s, \varepsilon) \right)$$

until T' ==T

This transitive closure algorithm terminates because there is a finite number of states in the NFA

DFAedge and NFA Simulation

Suppose we are in state DFA d = {s_i, s_k,s_l}

By moving with character c from d we reach a set of new NFA states, call these DFAedge(d,c), a new or already existing DFA state

$$DFAedge(d,c) = closure(\bigcup_{s \in d} edge(s,c))$$

NFA simulation:

let the input string be $c_1 \ldots c_k$

d=closure({s1}) // **s_1** the start state of the NFA

for i from 1 to k

d = DFAedge(d,c_i)

Constructing a DFA with closure and DFAEdge

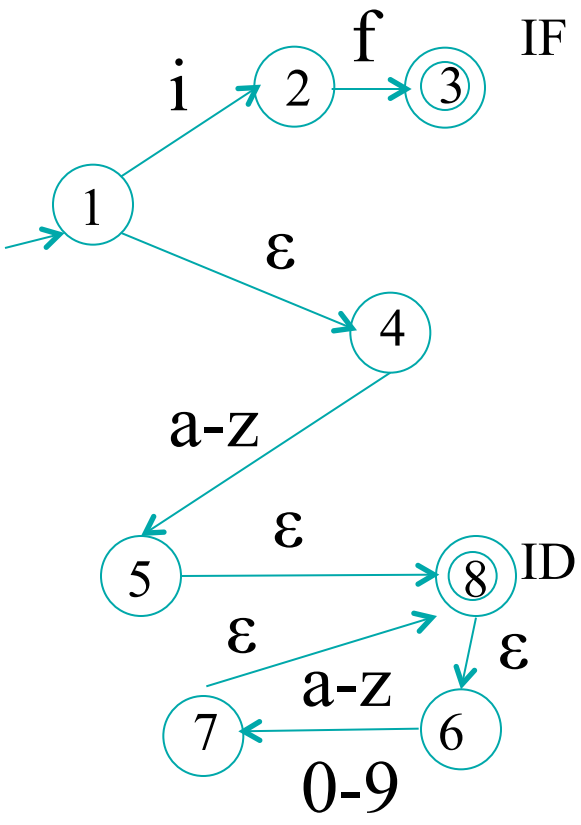
state d_1 = closure(s_1) the closure of the start state of the NFA

make new states by moving from existing states with a character c, using DFAEdge(d,c); record these in the transition table

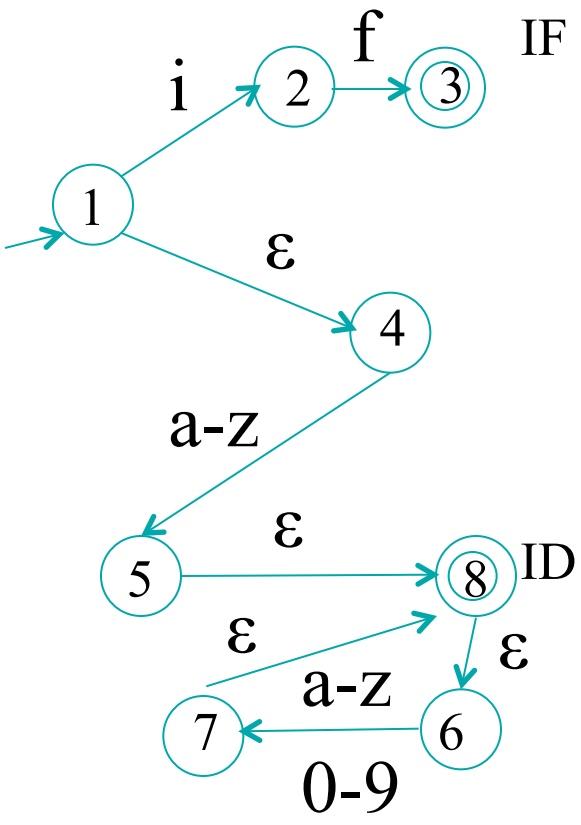
make accepts in the transition table, if there is an accepting state in d, decide priority if more than one accept state.

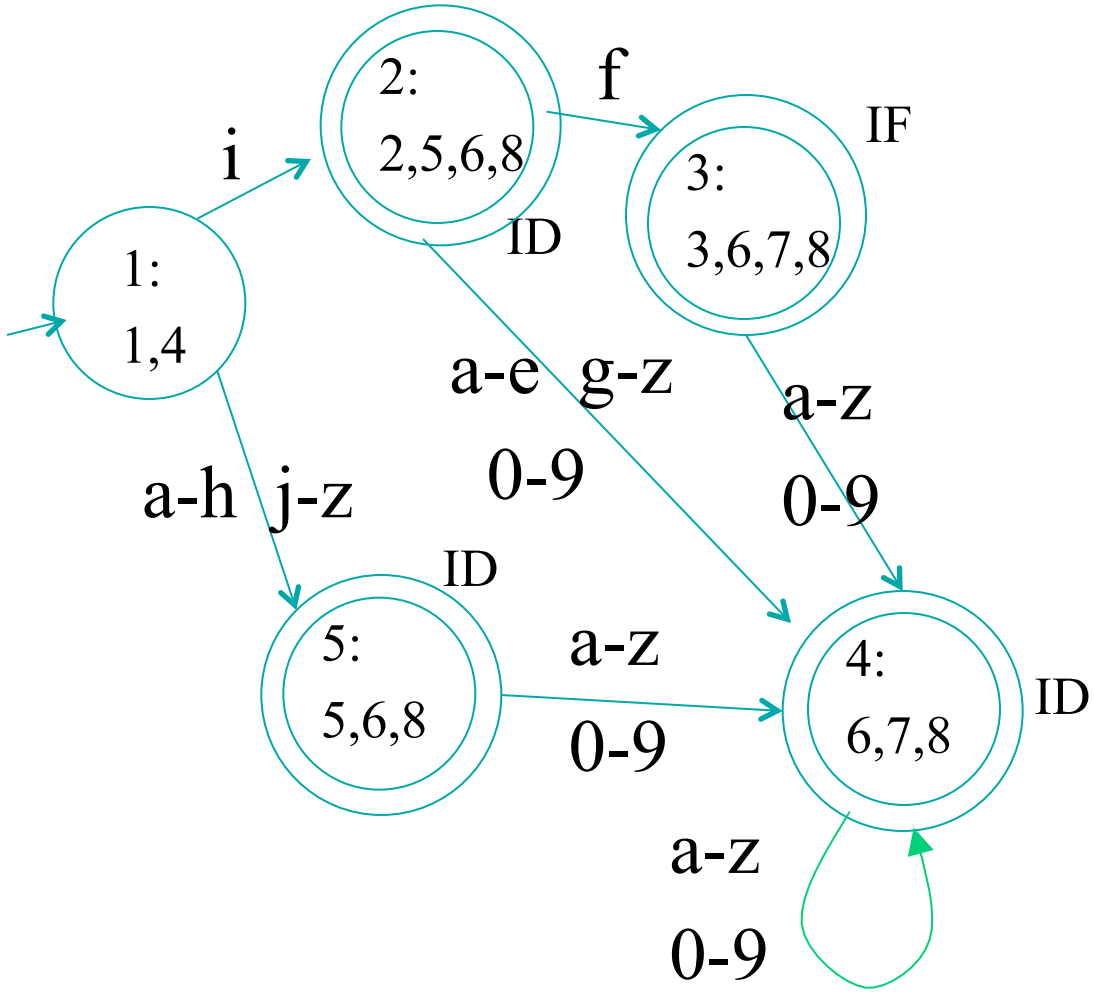
Instead of characters we use non-overlapping (DFA) character classes to keep the table manageable.

NFA to DFA
(let's build it)



NFA to DFA


1
i
2
f
3
IF
ε
4
a-z
5
ε
8
ID
ε
ε
a-z
7
6
0-9


1:
1,4
i
2:
2,5,6,8
ID
f
3:
3,6,7,8
IF
a-h j-z
a-e g-z
0-9
a-z
0-9
5:
5,6,8
ID
a-z
0-9
4:
6,7,8
ID
a-z
0-9

The transition table for IF ID

p	NFAstates(p)	i	f	a-h j-z	a-e,g-z 0-9	a-z,0-9	ACPT
1	{1,4}	{2,5,6,8}		{5,6,8}			
2	{2,5,6,8}		{3,6,7,8}		{6,7,8}		ID
3	{3,6,7,8}					{6,7,8}	IF
4	{6,7,8}					{6,7,8}	ID
5	{5,6,8}					{6,7,8}	ID

Suggested Exercise

Build an NFA and a DFA for integer and float literals

dot: “.”

dig: [0-9]

int-lit: dig$^+$

float-lit: dig* dot dig+