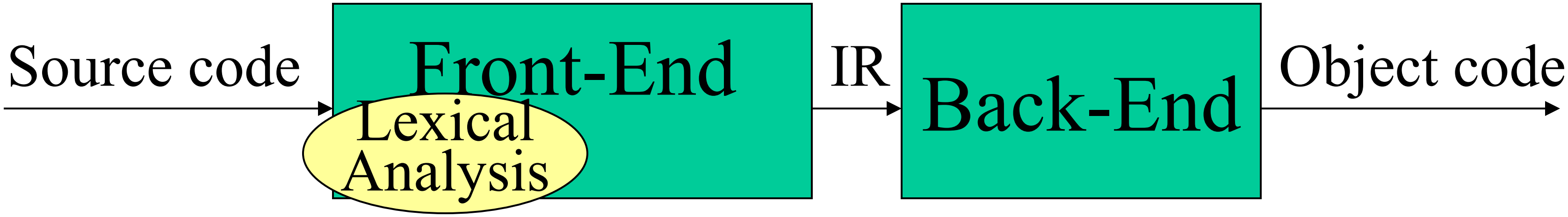
Lecture 4: Lexical Analysis II: From REs to DFAs



(from last lecture) Lexical Analysis:

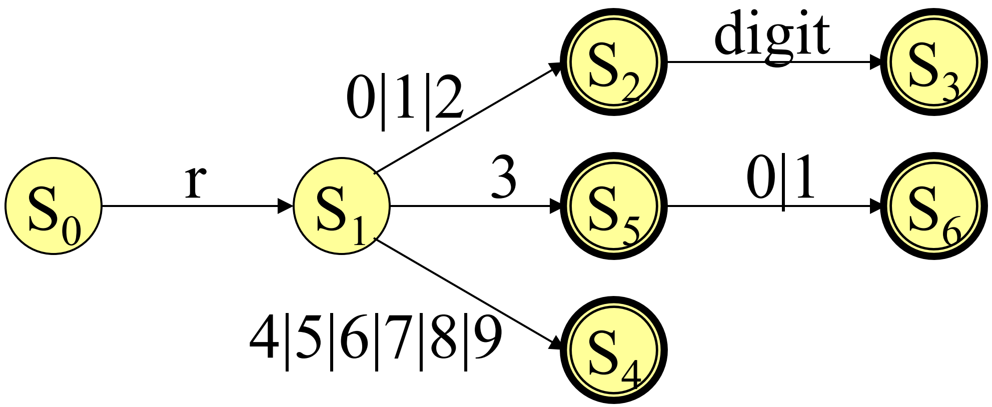
- Regular Expressions (REs) are formulae to describe a (regular) language.
- Every RE can be converted to a Deterministic Finite Automaton (DFA).
- DFAs can automate the construction of lexical analysers.

Today's lecture:

Algorithms to derive a DFA from a RE.

An Example (recognise r0 through r31)

$$Register \rightarrow r\ ((0|1|2)\ (Digit|\varepsilon)\ |\ (4|5|6|7|8|9)\ |\ (3|30|31))$$

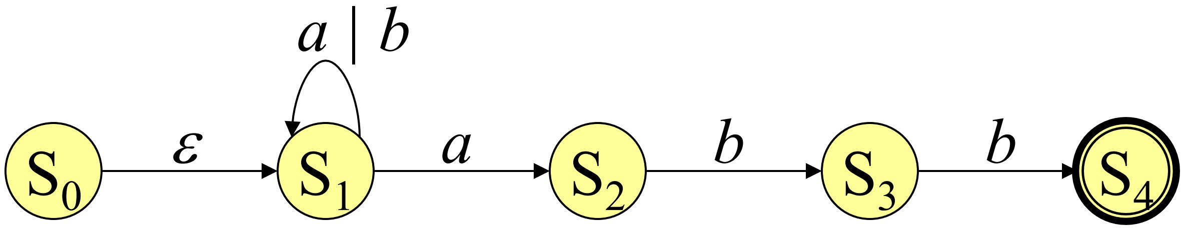


State	'r'	0,1	2	3	4,5,…,9
0	1	-	-	-	-
1	-	2	2	5	4
2(final)	-	3	3	3	3
3(final)	-	-	-	-	-
4(final)	-	-	-	-	-
5(final)	-	6	-	-	-
6(final)	-	-	-	-	-

- Same code skeleton (Lecture 3, slide 11) can be used!
- Different (bigger) transition table.
- Our Deterministic Finite Automaton (DFA) recognises only r0 through r31.

Non-deterministic Finite Automata

*What about a RE such as (a | b)*abb?*



- This is a Non-deterministic Finite Automaton (NFA):
  - S_0 has a transition on ε; S_1 has two transitions on *a* (not possible for a DFA).
- A DFA is a special case of an NFA:
  - for each state and each transition there is at most one rule.
- A DFA can be simulated with an NFA (obvious!)
- A NFA can be simulated with a DFA (less obvious).
  - Simulate sets of possible states.

Why study NFAs? DFAs can lead to faster recognisers than NFAs but may be much bigger. Converting a RE into an NFA is more direct.

The Big Picture:
Automatic Lexical Analyser Construction

To convert a specification into code:

- Write down the RE for the input language.
- Convert the RE to a NFA (Thompson's construction)
- Build the DFA that simulates the NFA (subset construction)
- Shrink the DFA (Hopcroft's algorithm)

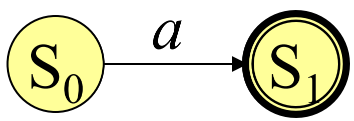
(for the curious: there is a full cycle - DFA to RE construction is all pairs, all paths)

Lexical analyser generators:

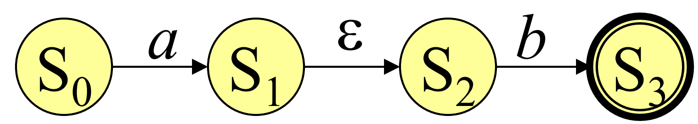
- lex or flex work along these lines.
- Algorithms are well-known and understood.
- Key issue is the interface to parser.

RE to NFA using Thompson's construction

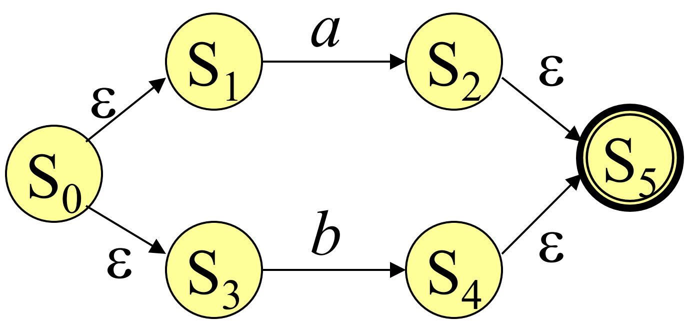
Key idea (Ken Thompson; CACM, 1968): NFA pattern for each symbol and/or operator: join them in precedence order.



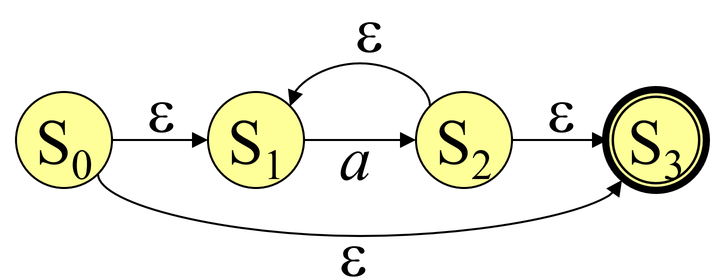
NFA for *a*



NFA for *ab*



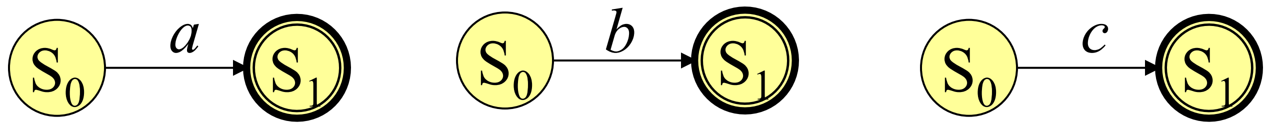
NFA for *a* | *b*

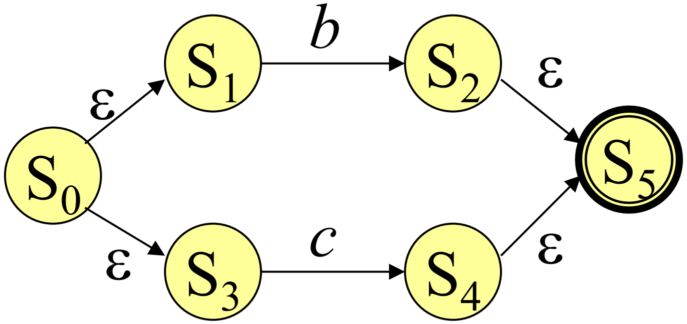


NFA for *a**

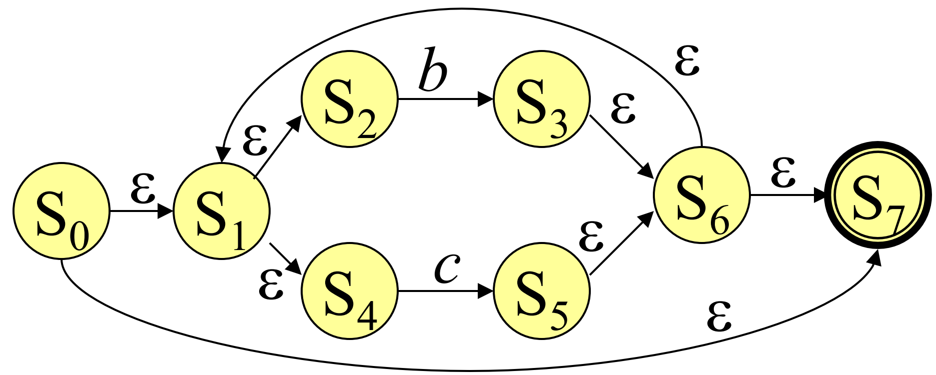
Example: Construct the NFA of *a (b|c)**

First: NFAs for *a, b, c*



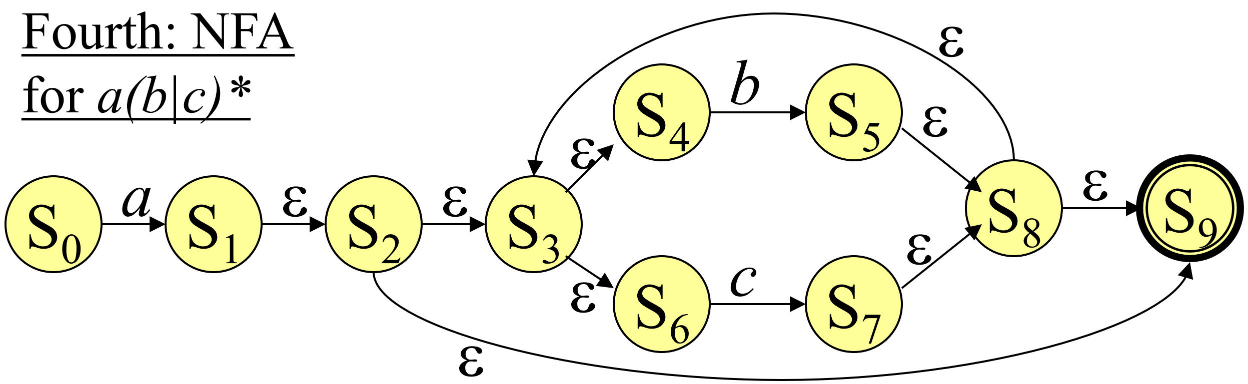


Second: NFA for *b|c*

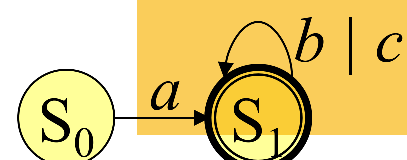


Third: NFA for *(b|c)**

Fourth: NFA for *a(b|c)**



Of course, a human would design a simpler one… But, we can automate production of the complex one...

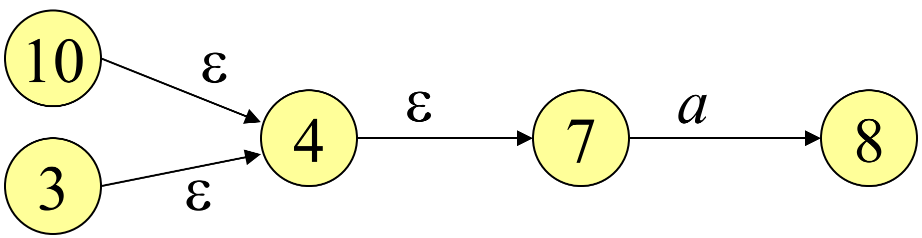


NFA to DFA: two key functions

- **move(s_i,a):** the (union of the) set of states to which there is a transition on input symbol **a** from state $\mathbf{s_i}$
- **ε-closure(s_i):** the (union of the) set of states reachable by **ε** from $\mathbf{s_i}$.

Example (see the diagram below):

- ε-closure(3)={3,4,7}; ε-closure({3,10})={3,4,7,10};
- move(ε-closure({3,10}),*a*)=8;



The Algorithm:

- start with the ε-closure of s_0 from NFA.
- Do for each unmarked state until there are no unmarked states:
  – for each symbol take their ε-closure(move(state,symbol))

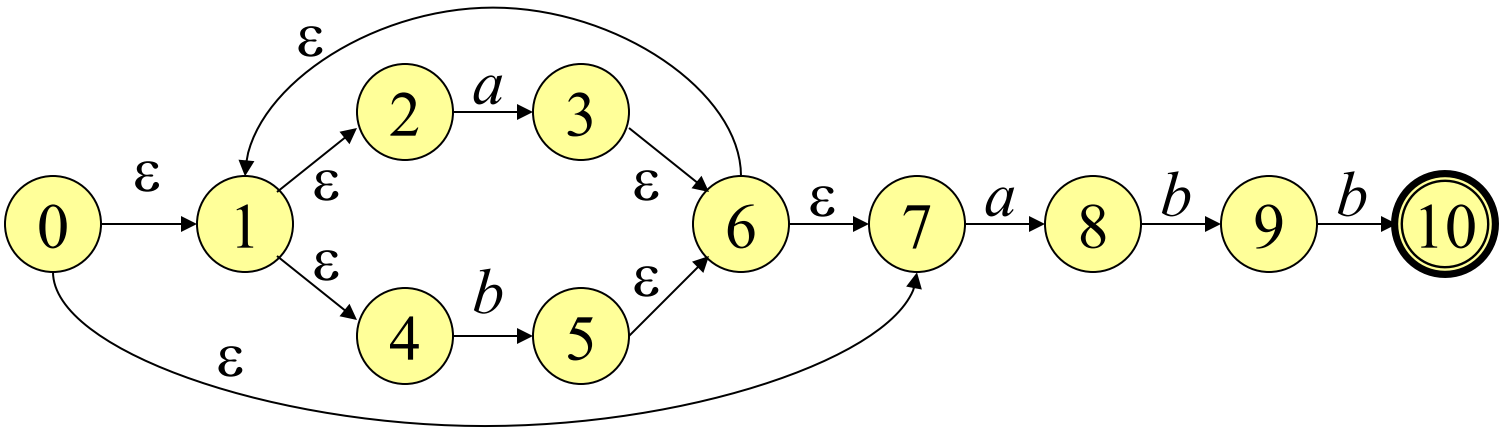
NFA to DFA with subset construction

Initially, ε-closure is the only state in Dstates and it is unmarked.
while there is an unmarked state T in Dstates
    mark T
    for each input symbol a
        U:=ε-closure(move(T,a))
        if U is not in Dstates then add U as unmarked to Dstates
        Dtable[T,a]:=U

- Dstates (set of states for DFA) and Dtable form the DFA.
- Each state of DFA corresponds to a set of NFA states that NFA could be in after reading some sequences of input symbols.
- This is a fixed-point computation.

It sounds more complex than it actually is!

Example: NFA for *(a | b)*abb*

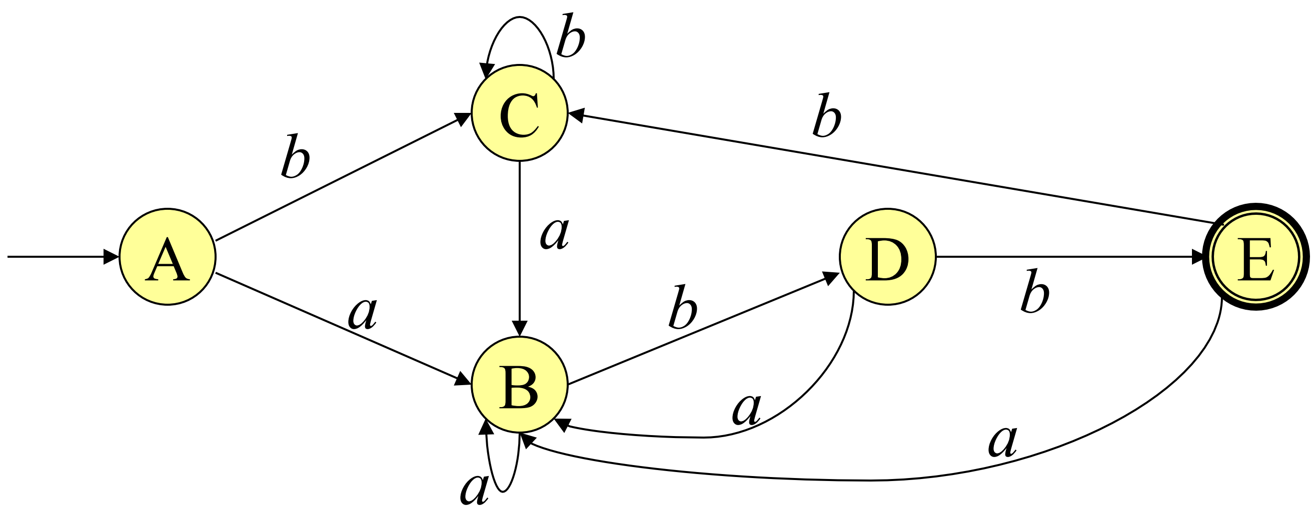


- A=ε-closure(0)={0,1,2,4,7}
- for each input symbol (that is, *a* and *b*):
  - B=ε-closure(move(A,*a*))=ε-closure({3,8})={1,2,3,4,6,7,8}
  - C=ε-closure(move(A,*b*))=ε-closure({5})={1,2,4,5,6,7}
  - Dtable[A,*a*]=B; Dtable[A,*b*]=C
- B and C are unmarked. Repeating the above we end up with:
  - C={1,2,4,5,6,7}; D={1,2,4,5,6,7,9}; E={1,2,4,5,6,7,10}; and
  - Dtable[B,*a*]=B; Dtable[B,*b*]=D; Dtable[C,*a*]=B; Dtable[C,*b*]=C; Dtable[D,*a*]=B; Dtable[D,*b*]=E; Dtable[E,*a*]=B; Dtable[E,*b*]=C; no more unmarked sets at this point!

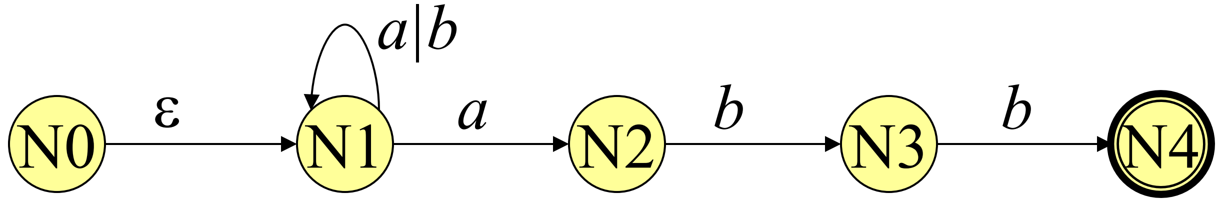
Result of applying subset construction

Transition table:

state	*a*	*b*
A	B	C
B	B	D
C	B	C
D	B	E
E(final)	B	C



Another NFA version of the same RE



Apply the subset construction algorithm:

Iteration	State	Contains	ε-closure(move(s,*a*))	ε-closure(move(s,*b*))
0	A	N0,N1	N1,N2	N1
1	B	N1,N2	N1,N2	N1,N3
	C	N1	N1,N2	N1
2	D	N1,N3	N1,N2	N1,N4
3	E	N1,N4	N1,N2	N1

Note:

- iteration 3 adds nothing new, so the algorithm stops.
- state E contains N4 (final state)

Enough theory… Let's conclude!

- We presented algorithms to construct a DFA from a RE.
- The DFA is not necessarily the smallest possible.
- Using an (automatically generated) transition table and the standard code skeleton (Lecture 3, slide 11) we can build a lexical analyser from regular expressions automatically. But, the size of the table can be large...
- Next time:
  - DFA minimisation; Practical considerations; Lexical Analysis wrap-up.
- Reading: Aho2 Sections 3.6-3.7; Aho1 pp. 113-125; Grune 2.1.6.1-2.1.6.6 (different style); Hunter 3.3 (very condensed); Cooper1 2.4-2.4.3