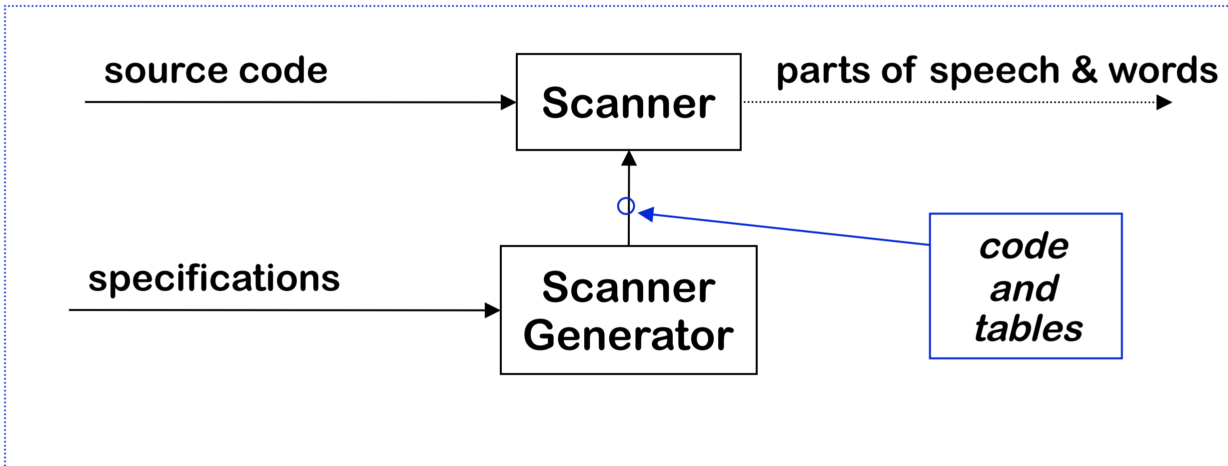
Lexical Analysis: Constructing a Scanner from Regular Expressions

Goal

- Show how to construct a FA to recognize any RE
- This Lecture
  - → Convert RE to an **nondeterministic finite automaton (NFA)**
    - Use Thompson's construction

Quick Review



Previous class:

- The scanner is the first stage in the front end
- Specifications can be expressed using regular expressions
- Build tables and code from a DFA

Register Name DFA Class Problem?

Consider the problem of recognizing register names

$$\textit{Register} \rightarrow r\ (0|1|2|\ \ldots\ |\ 9)\ (0|1|2|\ \ldots\ |\ 9)^*$$

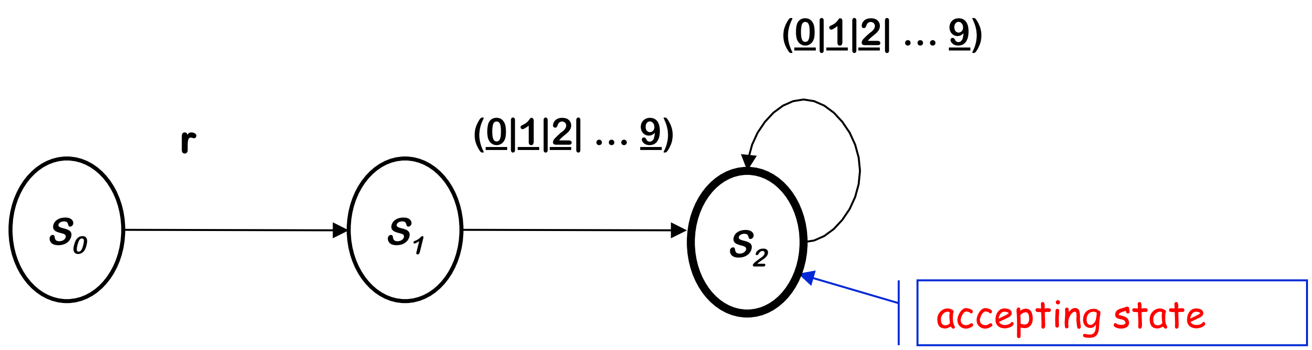
- Allows registers of arbitrary number
- Requires at least one digit

Register Name DFA Solution

Consider the problem of recognizing register names

Register $\rightarrow$ r (0|1|2| … | 9) (0|1|2| … | 9)*

RE corresponds to a recognizer (or DFA)

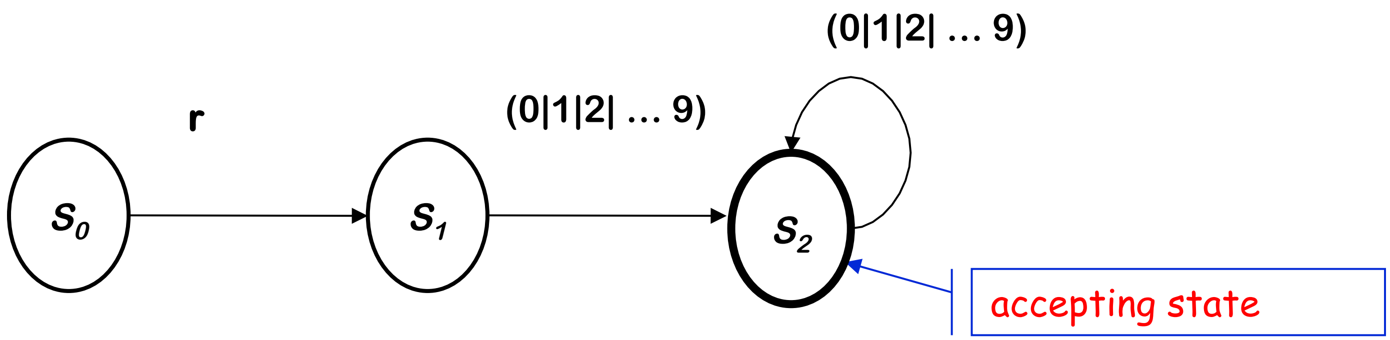


Recognizer for *Register*

Transitions on other inputs go to an error state, s_e

DFA operation

- Start in state S_0 & take transitions on each input character
- DFA accepts a word x *iff* x leaves it in a final state (S_2)



Recognizer for *Register*

So,

- r17 takes it through s_0, s_1, s_2 and accepts
- r takes it through s_0, s_1 and fails
- a takes it straight to s_e

Example

To be useful, recognizer must turn into code

```
Char ← next character
State ← s0

while (Char ≠ EOF)
    State ← δ(State,Char)
    Char ← next character

if (State is a final state )
    then report success
    else  report failure
```

Skeleton recognizer

δ	r	0,1,2,3,4, 5,6,7,8,9	*All others*
s_0	s_1	s_e	s_e
s_1	s_e	s_2	s_e
s_2	s_e	s_2	s_e
s_e	s_e	s_e	s_e

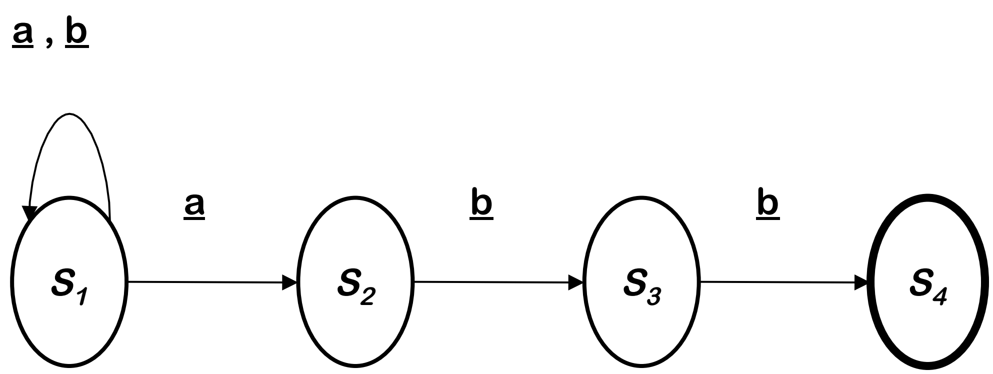
Table encoding RE

Non-deterministic Finite Automata

Each RE corresponds to a *deterministic finite automaton* (DFA)

- May be hard to directly construct the right DFA

For example, consider the RE (a | b)* abb.

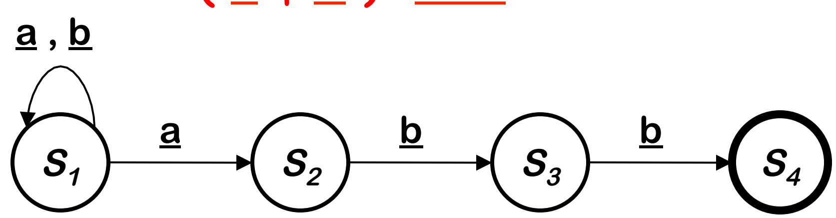


Non-deterministic Finite Automata

Each RE corresponds to a *deterministic finite automaton* (DFA)

- May be hard to directly construct the right DFA

What about an RE such as $(\underline{a} \mid \underline{b})^* \underline{abb}$?



This is a little different from typical DFAs!

- S_1 has two transitions on a

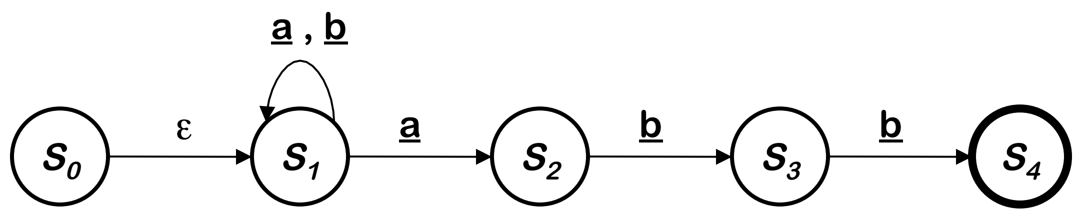
This is a *non-deterministic finite automaton* (NFA)

Non-deterministic Finite Automata

Each RE corresponds to a *deterministic finite automaton* (DFA)

- May be hard to directly construct the right DFA

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



This is a little different from typical DFAs!

- *S_1 has two transitions on a*
- S_0 has a transition on ε

This is a *non-deterministic finite automaton* (NFA)

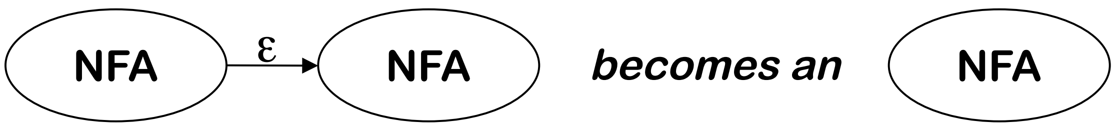
Nondeterministic Finite Automata

- An NFA accepts a string *x*

  iff ∃ a path though the graph from s_0 to a final state such that the edge labels spell *x*
- Transitions on ε consume no input
- To "run" the NFA, start in s_0 and *guess* the right transition at each choice point with multiple possibilities
  - → Always guess correctly
  - → If some sequence of correct guesses accepts x then accept

Why study NFAs?

- They are the key to automating the RE→DFA construction
- We can paste together NFAs with ε-transitions



Relationship between NFAs and DFAs

DFA is a special case of an NFA

- DFA has no ε transitions
- DFA's transition function is single-valued
- Same rules will work

DFA can be simulated with an NFA

→ *Obviously*

Relationship between NFAs and DFAs

NFA can be simulated with a DFA *(less obvious)*

- Simulate sets of possible states
- Possible exponential blowup in the state space
- Still, one state per character in the input stream

Subset construction builds a **DFA** that simulates an **NFA**.

Automating Scanner Construction

To convert a specification into code:

1 Write down the RE for the input language

2 Build a big NFA

3 Build the DFA that simulates the NFA

4 Systematically shrink the DFA

5 Turn it into code

Scanner generators

- Lex, Flex, and JLex work along these lines
- Algorithms are well-known and well-understood
- Key issue is interface to parser *(define all parts of speech)*

Automating Scanner Construction

RE→ NFA *(Thompson's construction)*

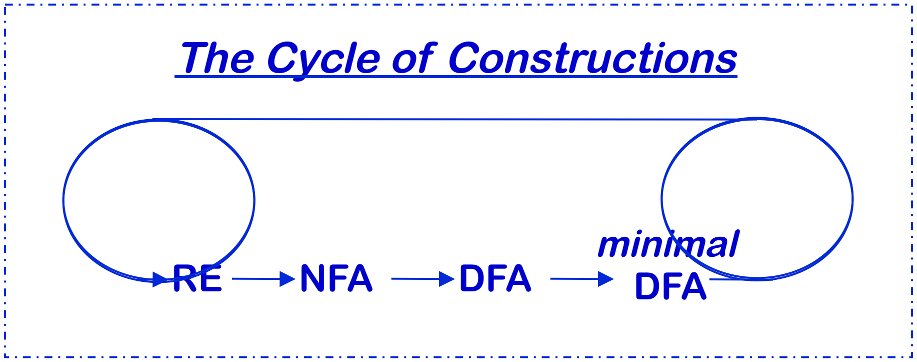
- Build an NFA for each term
- Combine them with ε-transitions

NFA → DFA *(Subset construction)*

- Build the simulation

DFA → Minimal DFA

- Hopcroft's algorithm



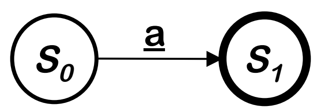
DFA →RE *(Not part of the scanner construction)*

- All pairs, all paths problem
- Take the union of all paths from s_0 to an accepting state

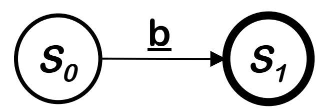
RE →NFA using Thompson's Construction

Key idea

- NFA pattern for each symbol and each operator
- Join them with ε transitions in precedence order

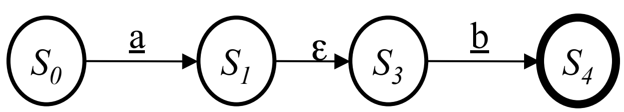


NFA for a



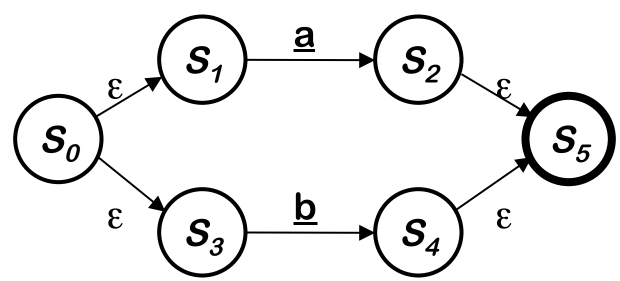
NFA for b

Concatenation



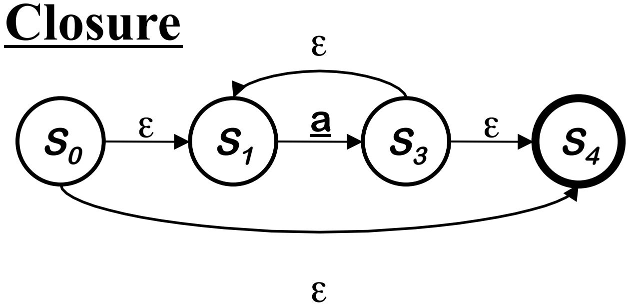
NFA for ab

Alternation



NFA for a | b

Closure

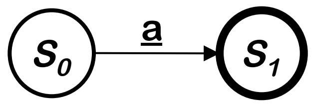


NFA for a*

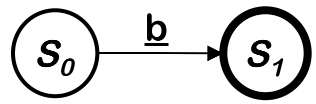
Ken Thompson, CACM, 1968

RE →NFA using Thompson's Construction

Let's try: a (b | c)*

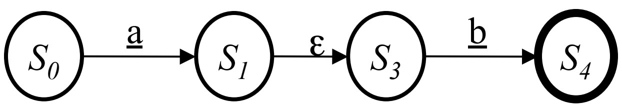


NFA for a



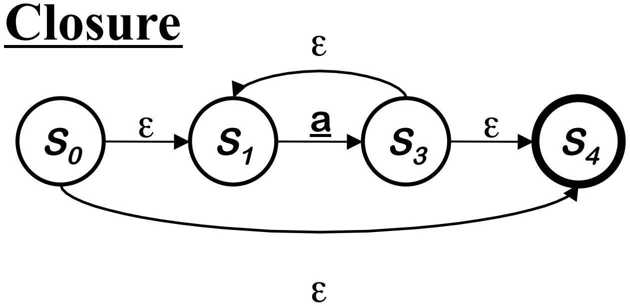
NFA for b

Concatenation



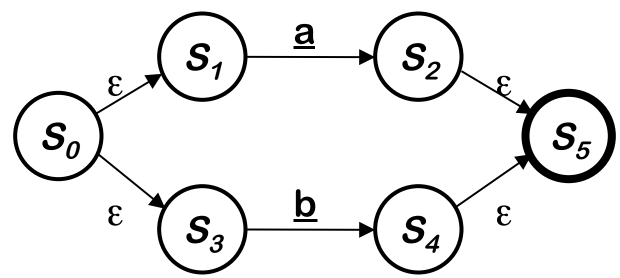
NFA for ab

Closure



NFA for a*

Alternation

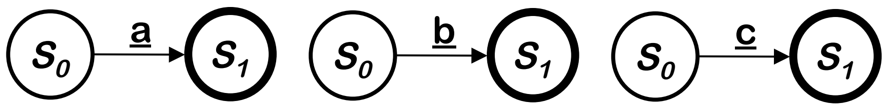


NFA for a | b

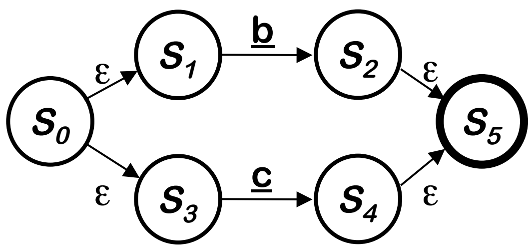
Example of Thompson's Construction

Let's try $\underline{a}\ (\ \underline{b}\ |\ \underline{c}\ )^*$

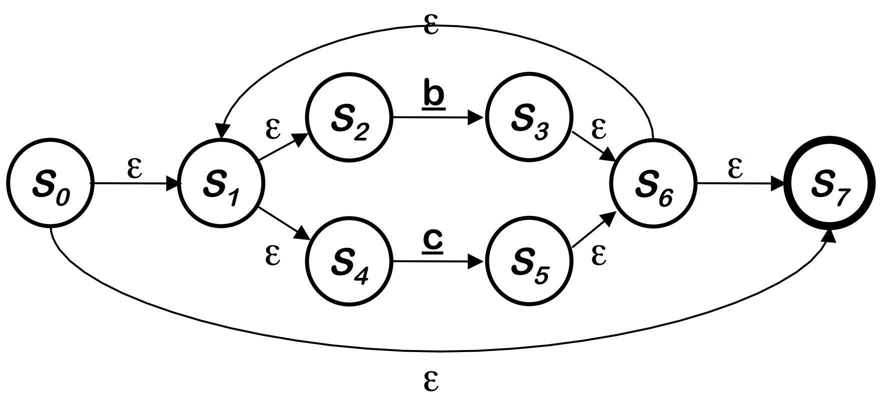
1. $\underline{a}$, $\underline{b}$, $\underline{c}$



2. $\underline{b}\ |\ \underline{c}$

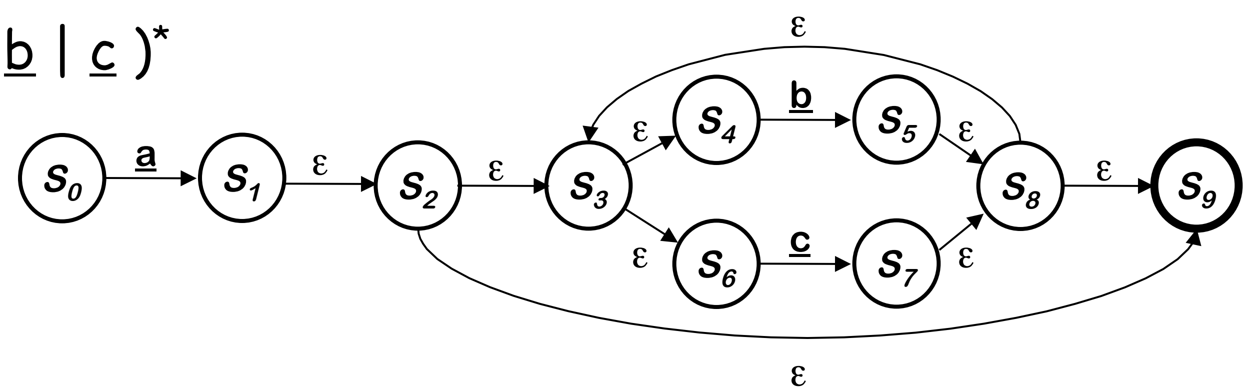


3. $(\ \underline{b}\ |\ \underline{c}\ )^*$

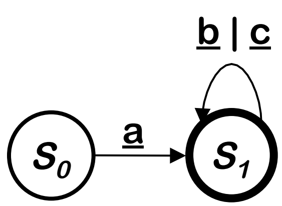


Example of Thompson's Construction *(cont'd)*

4. $\underline{a}\ (\ \underline{b}\ |\ \underline{c}\ )^*$



Of course, a human would design something simpler ...



But, we can automate production of the more complex one ...