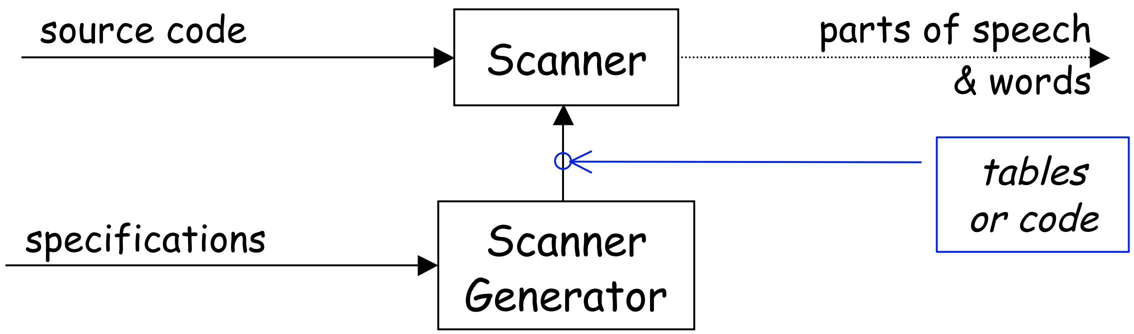
Lexical Analysis — Part II: Constructing a Scanner from Regular Expressions

COMP 412
Fall 2005

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

More Regular Expressions

- All strings of 1s and 0s ending in a 1

$$(0 \mid 1)^* \, 1$$

- All strings over lowercase letters where the vowels (a,e,i,o, & u) occur exactly once, in ascending order

$$\mathit{Cons} \rightarrow (b \mid c \mid d \mid f \mid g \mid h \mid j \mid k \mid l \mid m \mid n \mid p \mid q \mid r \mid s \mid t \mid v \mid w \mid x \mid y \mid z)$$

$$\mathit{Cons}^* \; a \; \mathit{Cons}^* \; e \; \mathit{Cons}^* \; i \; \mathit{Cons}^* \; o \; \mathit{Cons}^* \; u \; \mathit{Cons}^*$$

- All strings of 1s and 0s that do not contain three 0s in a row:

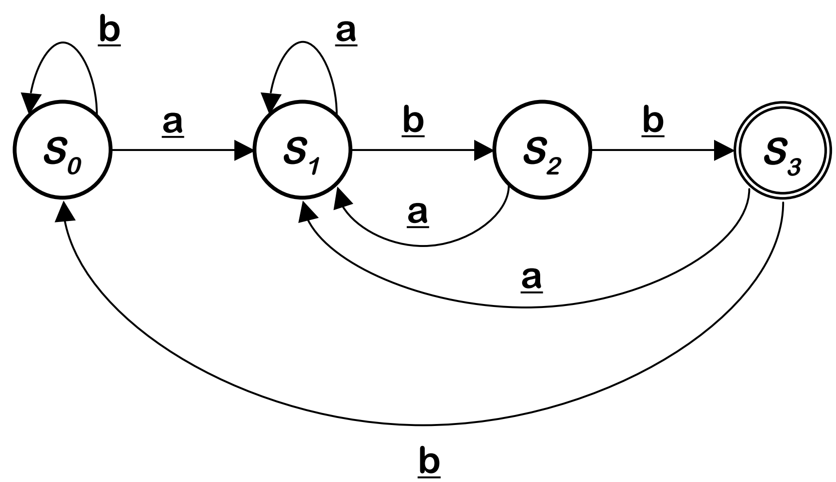
$$(1^* (\varepsilon \mid 01 \mid 001) \, 1^*)^* (\varepsilon \mid 0 \mid 00)$$

Goal

- We will show how to construct a finite state automaton to recognize any RE
- Overview:
  - Direct construction of a nondeterministic finite automaton (NFA) to recognize a given RE
    - Requires ε-transitions to combine regular subexpressions
  - Construct a deterministic finite automaton (DFA) to simulate the NFA
    - Use a set-of-states construction
  - Minimize the number of states in the DFA
    - Hopcroft state minimization algorithm
  - Generate the scanner code
    - Additional specifications needed for the actions

Non-deterministic Finite Automata

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

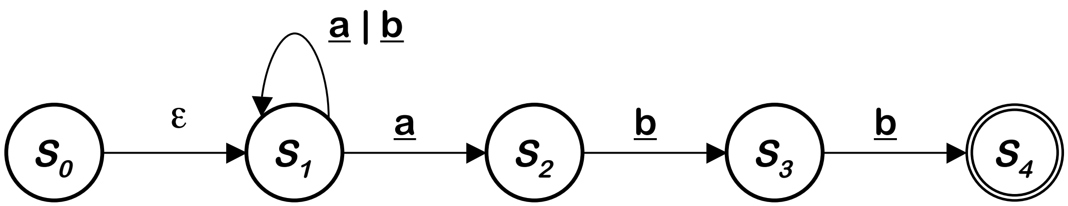


Each RE corresponds to a *deterministic finite automaton* (DFA)

- May be hard to directly construct the right DFA

Non-deterministic Finite Automata

Here is another RE for (a | b)* abb



This recognizer has different properties

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

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

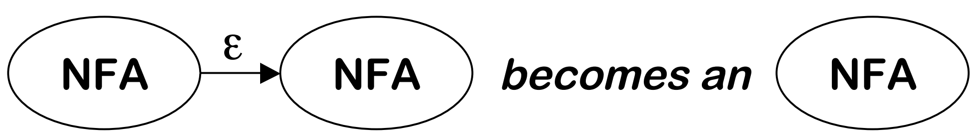
Non-deterministic Finite Automata

An NFA accepts a string *x iff* ∃ a path though the transition graph from s_0 to a final state such that the edge labels spell *x*, ignoring ε's

- Transitions on ε consume no input
- To "run" the NFA, start in s_0 and *guess* the right transition at each step
  - 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*

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

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 and Flex work along these lines
- Algorithms are well-known and well-understood
- Key issue is interface to parser *(define all parts of speech)*
- You could build one in a weekend!

Automating Scanner Construction

RE→ NFA *(Thompson's construction)*

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

NFA → DFA *(subset construction)*

- Build the simulation

DFA → Minimal DFA

- Hopcroft's algorithm

The Cycle of Constructions

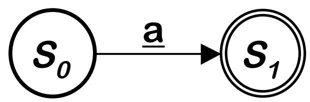
DFA →RE *(Not part of the scanner construction)*

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

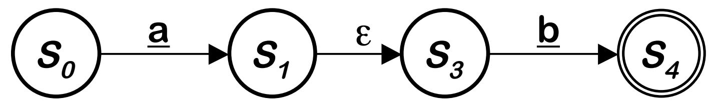
RE →NFA using Thompson's Construction

Key idea

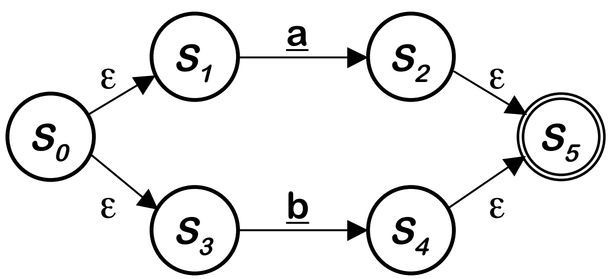
- NFA pattern for each symbol & each operator
- Join them with ε moves in precedence order



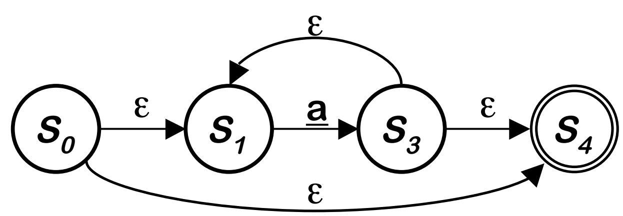
NFA for a



NFA for ab



NFA for a | b



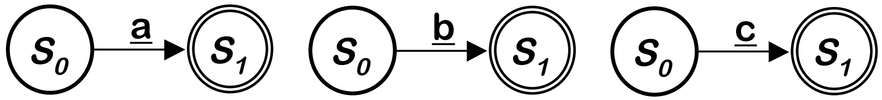
NFA for a*

Ken Thompson, CACM, 1968

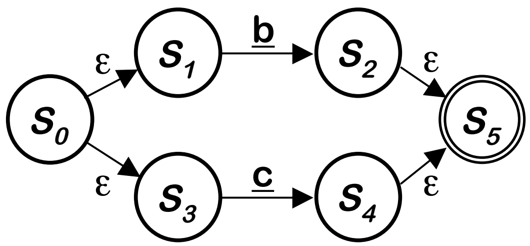
Example of Thompson's Construction

Let's try a (b | c)*

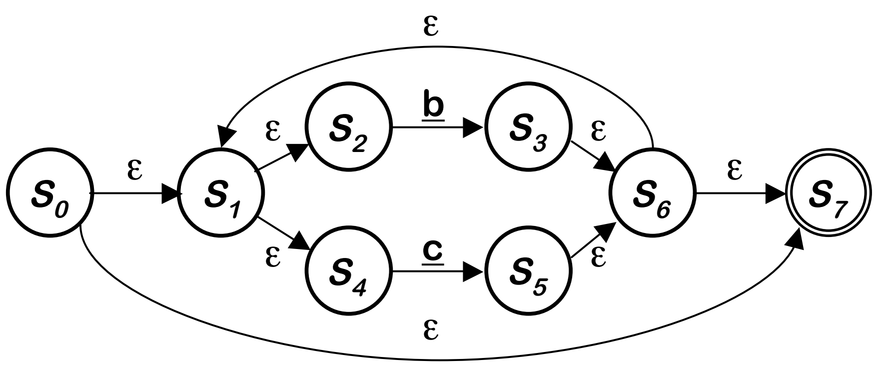
1. a, b, & c



2. b | c

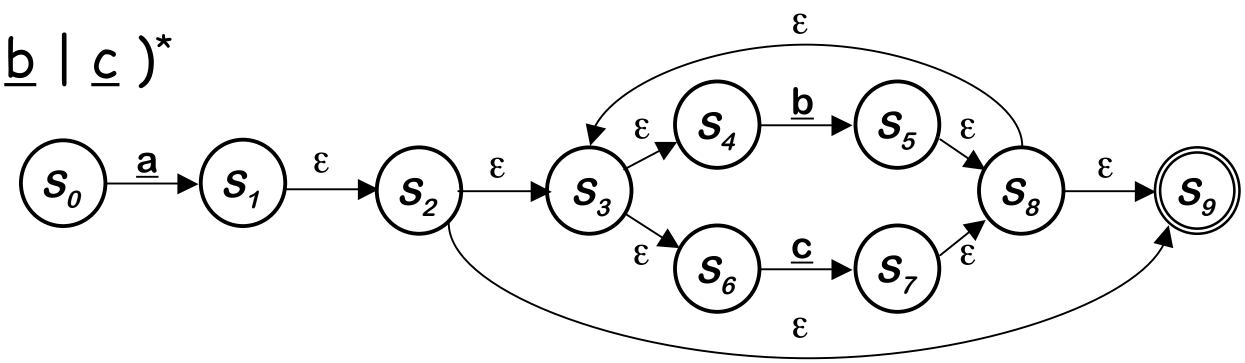


3. (b | c)*

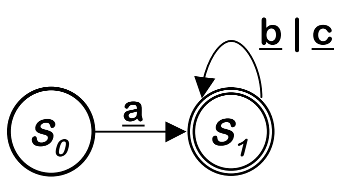


Example of Thompson's Construction *(con't)*

4. a (b | c)*



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



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

Automating Scanner Construction

RE→ NFA *(Thompson's construction)* ✓

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

NFA → DFA *(subset construction)* ⇐

- Build the simulation

DFA → Minimal DFA

- Hopcroft's algorithm

The Cycle of Constructions

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

NFA →DFA with Subset Construction

Need to build a simulation of the NFA

Two key functions

- *Move*(s_i , $\underline{a}$) is the set of states reachable from s_i by $\underline{a}$
- *ε-closure*(s_i) is the set of states reachable from s_i by ε

The algorithm:

- Start state derived from s_0 of the NFA
- Take its ε-closure S_0 = ε-closure($\{s_0\}$)
- Take the image of S_0, Move(S_0, α) for each $\alpha \in \Sigma$, and take its ε-closure
- Iterate until no more states are added

Sounds more complex than it is…

NFA →DFA with Subset Construction

The algorithm:

$s_0 \leftarrow \varepsilon\text{-}closure(\{n_0\})$
$S \leftarrow \{ s_0 \}$
$W \leftarrow \{ s_0 \}$
while ($W \neq \emptyset$)
    select and remove s from W
    for each $\alpha \in \Sigma$
        $t \leftarrow \varepsilon\text{-}closure(Move(s,\alpha))$
        $T[s,\alpha] \leftarrow t$
        if ($t \notin S$) *then*
            add t to S
            add t to W

Let's think about why this works

The algorithm halts:

1. *S* contains no duplicates (test before adding)
2. $2^{\{\text{NFA states}\}}$ is finite
3. while loop adds to *S*, but does not remove from *S (monotone)*

⇒ the loop halts

S contains all the reachable NFA states

It tries each character in each s_i.

It builds every possible NFA configuration.

⇒ *S and T form the DFA*

This test is a little tricky

NFA →DFA with Subset Construction

The algorithm:

$s_0 \leftarrow \varepsilon\text{-}closure(\{n_0\})$
$S \leftarrow \{ s_0 \}$
$W \leftarrow \{ s_0 \}$
while ($W \neq \emptyset$)
   select and remove s from W
   for each $\alpha \in \Sigma$
      $t \leftarrow \varepsilon\text{-}closure(Move(s,\alpha))$
      $T[s,\alpha] \leftarrow t$
      if ($t \notin S$) *then*
         add t to S
         add t to W

Let's think about why this works

The algorithm halts:

1. *S* contains no duplicates (test before adding)
2. $2^{\{\text{NFA states}\}}$ is finite
3. while loop adds to *S*, but does not remove from *S (monotone)*

⇒ the loop halts

S **contains all the reachable NFA states**

It tries each character in each s_i.

It builds every possible NFA configuration.

⇒ *S and T form the DFA*

Any DFA state containing an NFA final state becomes a DFA final state.

NFA →DFA with Subset Construction

Example of a *fixed-point* computation

- Monotone construction of some finite set
- Halts when it stops adding to the set
- Proofs of halting & correctness are similar
- These computations arise in many contexts

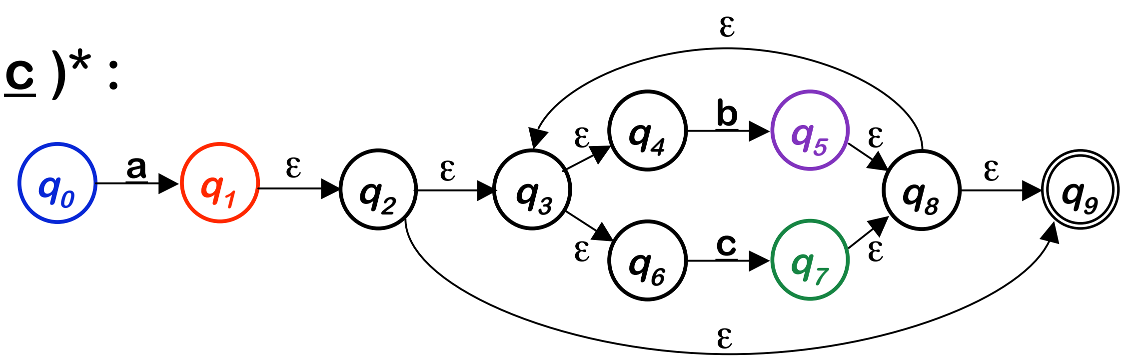
Other fixed-point computations

- Canonical construction of sets of LR(1) items
  - Quite similar to the subset construction
- Classic data-flow analysis (& Gaussian Elimination)
  - Solving sets of simultaneous set equations

We will see many more fixed-point computations

NFA →DFA with Subset Construction

a (b | c)* :

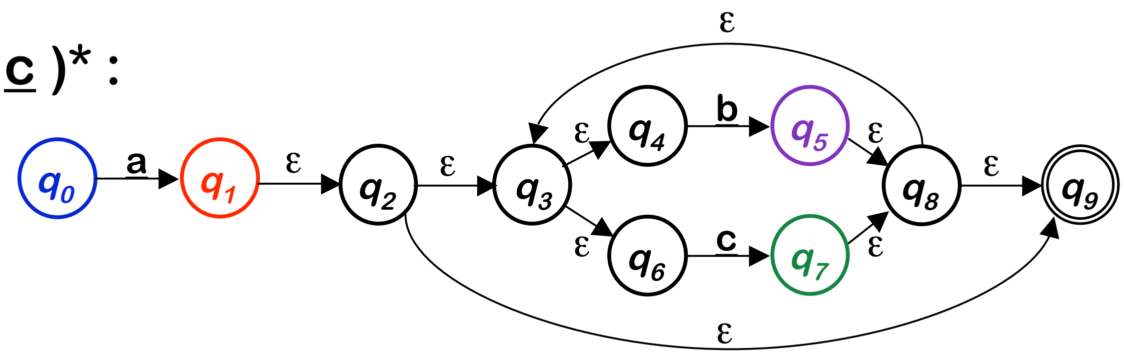


		ε-closure(Move(s,*))		
	NFA states	a	b	c
s_0	q_0			

Final states

NFA →DFA with Subset Construction

a (b | c)* :

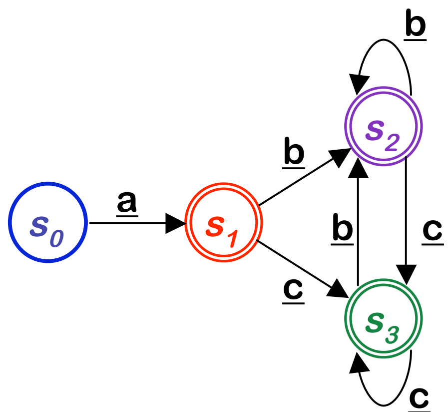


		ε-closure(Move(s,*))		
	NFA states	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	*none*
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	*none*	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	*none*	s_2	s_3

Final states

NFA →DFA with Subset Construction

The DFA for a (b | c)*



δ	a	b	c
s_0	s_1	-	-
s_1	-	s_2	s_3
s_2	-	s_2	s_3
s_3	-	s_2	s_3

- Ends up smaller than the NFA
- All transitions are deterministic
- Use same code skeleton as before

ε-transitions mess up the cost model, anyway

Where are we? Why are we doing this?

RE → NFA *(Thompson's construction)* ✓

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

NFA → DFA *(subset construction)* ✓

- Build the simulation

DFA → Minimal DFA ←

- Hopcroft's algorithm

DFA → RE

- All pairs, all paths problem
- Union together paths from s_0 to a final state

The Cycle of Constructions

Enough theory for today