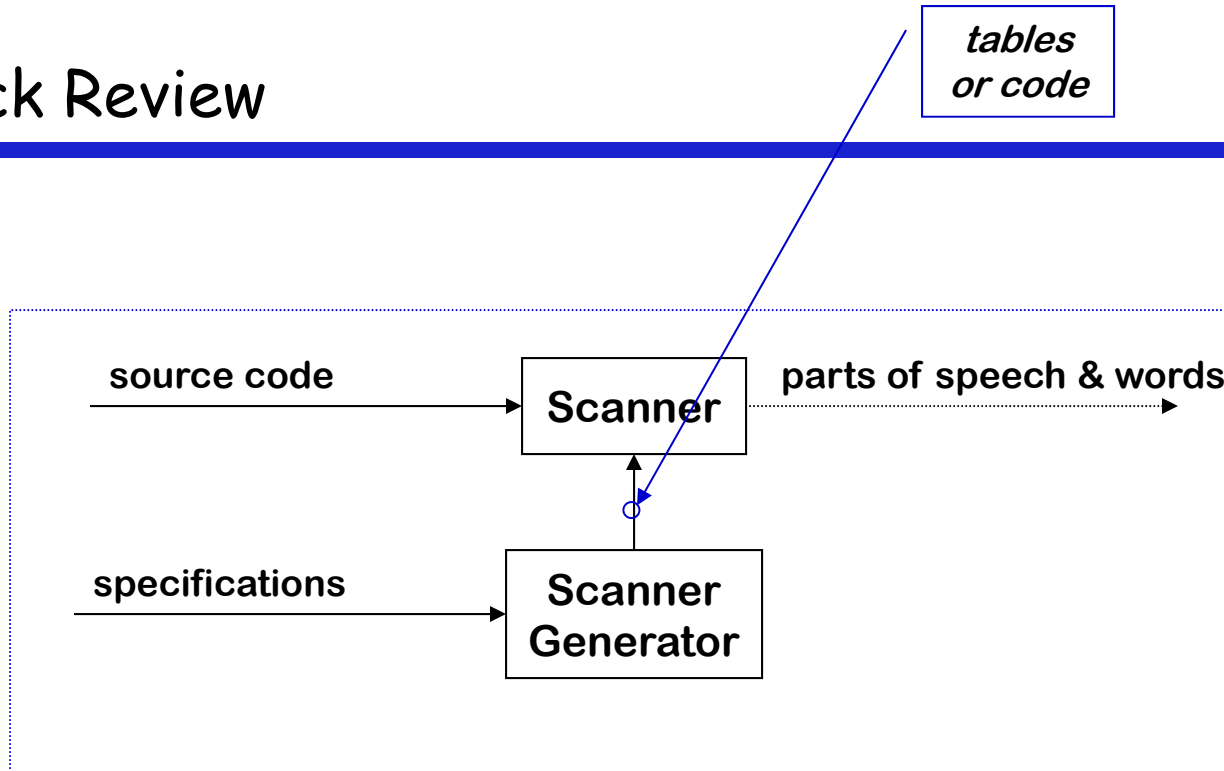


Lexical Analysis — Part II: Constructing a Scanner from Regular Expressions

Copyright 2003, Keith D. Cooper, Ken Kennedy & Linda Torczon, all rights reserved.
Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.

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

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
 - Hopcroft state minimization algorithm
 - Generate the scanner code
 - Additional specifications needed for details

More Regular Expressions

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

$(\underline{0} | \underline{1})^* \underline{1}$

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

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

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

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

More Regular Expressions

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

$(\underline{0} | \underline{1})^* \underline{1}$

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

$Cons \rightarrow (\underline{b} | \underline{c} | \underline{d} | \underline{f} | \underline{g} | \underline{h} | \underline{j} | \underline{k} | \underline{l} | \underline{m} | \underline{n} | \underline{p} | \underline{q} | \underline{r} | \underline{s} | \underline{t} | \underline{v} | \underline{w} | \underline{x} | \underline{y} | \underline{z})$
 $Cons^* \underline{a} Cons^* \underline{e} Cons^* \underline{i} Cons^* \underline{o} Cons^* \underline{u} Cons^*$

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

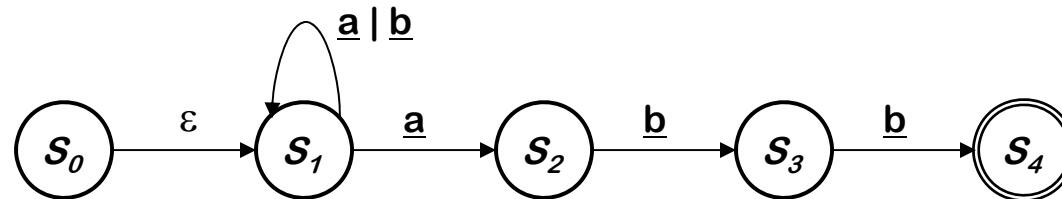
$(\underline{1}^* (\varepsilon | \underline{01} | \underline{001}) \underline{1}^*)^* (\varepsilon | \underline{0} | \underline{00})$

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 $(\underline{a} \mid \underline{b})^* \underline{a} \underline{b} \underline{b}$?



This is a little different

- S_0 has a transition on ϵ
- S_1 has two transitions on \underline{a}

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

Non-deterministic Finite Automata

- An NFA accepts a string x iff \exists a path through the transition 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 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 \rightarrow 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 \rightarrow NFA (Thompson's construction)

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

NFA \rightarrow DFA (subset construction)

- Build the simulation

DFA \rightarrow Minimal DFA

- Hopcroft's algorithm

DFA \rightarrow 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

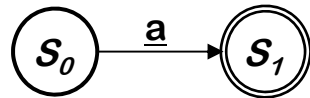
The Cycle of Constructions



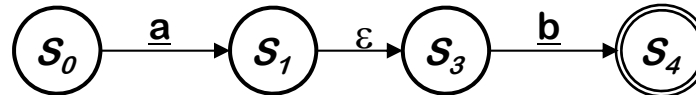
RE \rightarrow NFA using Thompson's Construction

Key idea

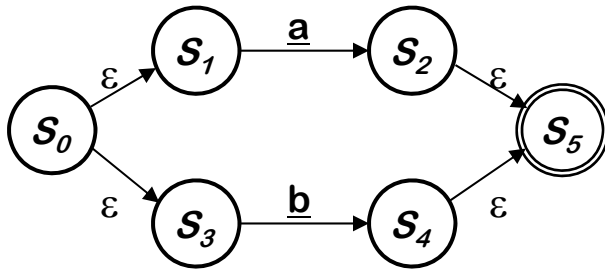
- NFA pattern for each symbol & each operator
- Join them with ϵ moves in precedence order



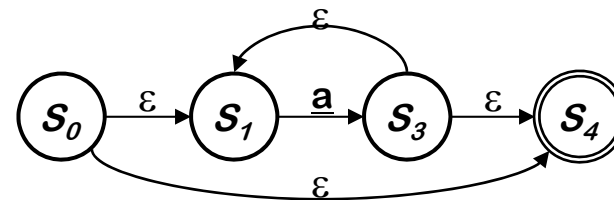
NFA for a



NFA for ab



NFA for a | b



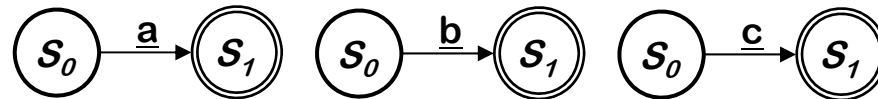
NFA for a*

Ken Thompson, CACM, 1968

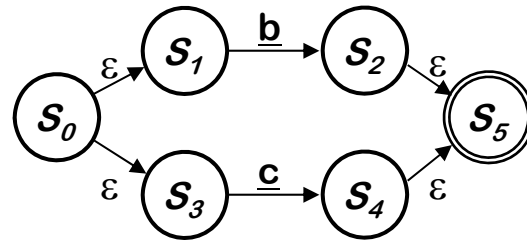
Example of Thompson's Construction

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

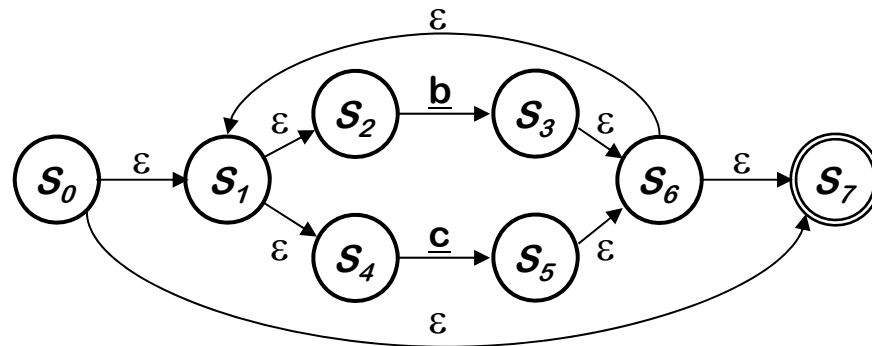
1. \underline{a} , \underline{b} , & \underline{c}



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

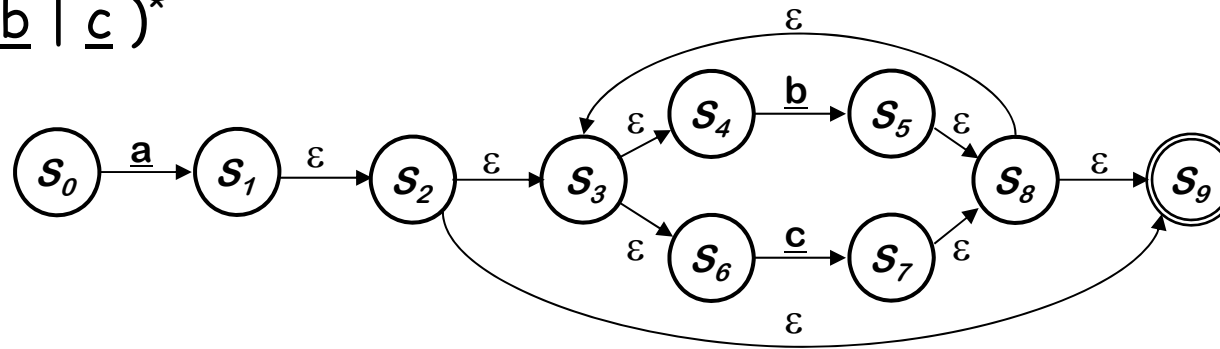


3. $(\underline{b} \mid \underline{c})^*$

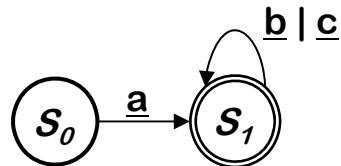


Example of Thompson's Construction (con't)

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



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



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

NFA \rightarrow DFA with Subset Construction

Need to build a simulation of the NFA

Two key functions

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

The algorithm:

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

Sounds more complex than it is...

NFA \rightarrow DFA with Subset Construction

The algorithm:

$s_0 \leftarrow \varepsilon\text{-closure}(q_{0n})$
while (S is still changing)
 for each $s_i \in S$
 for each $\alpha \in \Sigma$
 $s_j \leftarrow \varepsilon\text{-closure}(\text{Move}(s_i, \alpha))$
 if ($s_j \notin S$ *) then*
 add s_j *to* S *as* s_j
 $T[s_i, \alpha] \leftarrow s_j$

Let's think about why this works

The algorithm halts:

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

\Rightarrow the loop halts

S contains all the reachable NFA states

It tries each character in each s_i .

It builds every possible NFA configuration.

$\Rightarrow S$ and T form the DFA

NFA \rightarrow 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

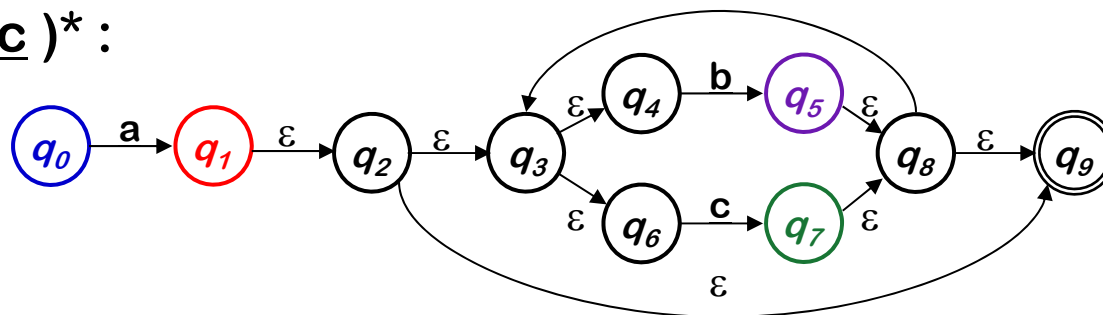
Other fixed-point computations

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

We will see many more fixed-point computations

NFA → DFA with Subset Construction

$a(b|c)^*$:



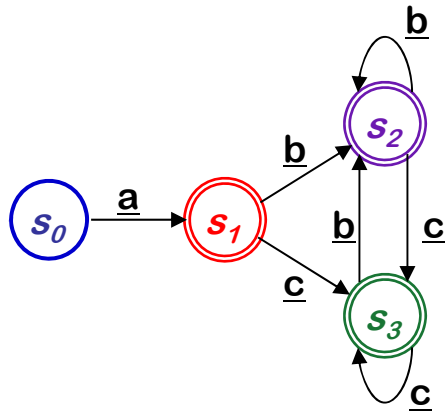
Applying the subset construction:

		ϵ -closure (move(s,*))		
	NFA states	<u>a</u>	<u>b</u>	<u>c</u>
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 \rightarrow DFA with Subset Construction

The DFA for $\underline{a}(\underline{b} \mid \underline{c})^*$



δ	<u>a</u>	<u>b</u>	<u>c</u>
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

Where are we? Why are we doing this?

RE \rightarrow NFA (Thompson's construction) 

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

NFA \rightarrow DFA (subset construction) 

- Build the simulation

DFA \rightarrow Minimal DFA

- Hopcroft's algorithm

DFA \rightarrow RE

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

The Cycle of Constructions



Enough theory for today