Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 10

Simplifying Finite Automata, and Lexical Analysis

slides by Graham Farr
based in part on previous slides by David Albrecht

Overview

- Simplifying Finite Automata
- Implementing Finite Automata
- Lexical Analyzer
- Tokens and lexemes

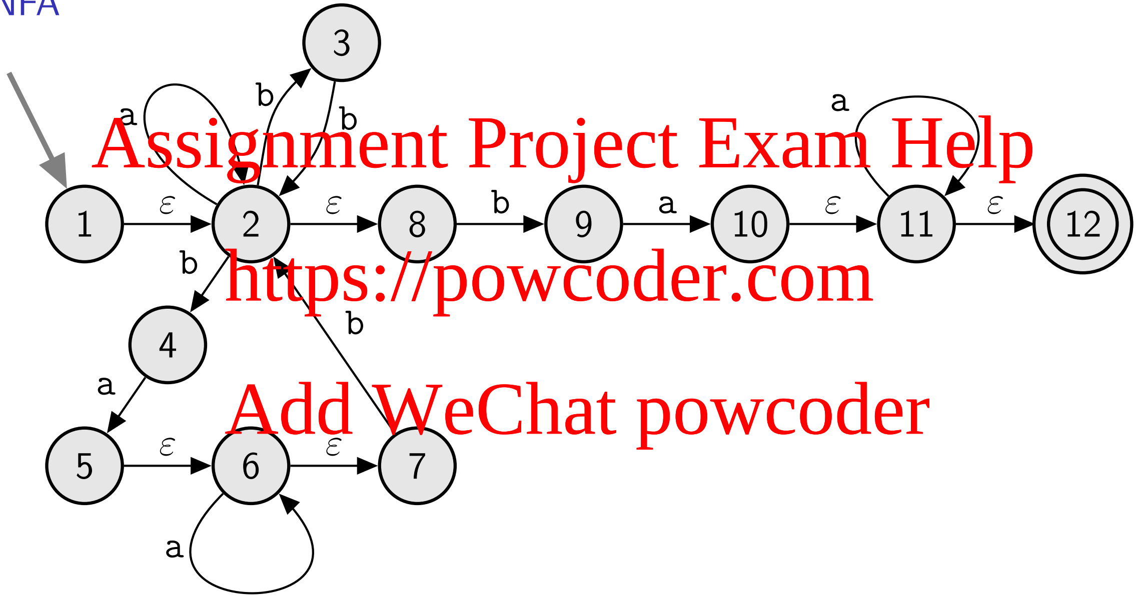
Matching a Regular Expression

Write a program which reads in a character string, over alphabet {a,b}, one character at a time and identifies whether or not the string matches the following regular expression.

$$(\mathtt{a} \cup \mathtt{bb} \cup \mathtt{baa}^*\mathtt{b})^*\mathtt{baa}^*$$

1. Convert regular expression to NFA.
2. Convert NFA to DFA.
3. **Simplify DFA.**

NFA



DFA

		a	b
Start	{1,2,8}	{2,8}	{3,4,9}
	{2,8}	{2,8}	{3,4,9}
	{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final	{5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final	{6,7,11,12}	{6,7,11,12}	{2,8}

A **Final State** and a **non-Final State** are *fundamentally different*.

They *cannot be combined*.

So:

- Give all **Final States** one colour.
- Give all **non-Final States** a *different* colour.

Different colours $\implies$ *different* states.

- They *cannot* be combined.

Same colours $\not\Longrightarrow$ same states.

- The states *may or may not* be combined.
- We have not yet ruled out combining them.

DFA

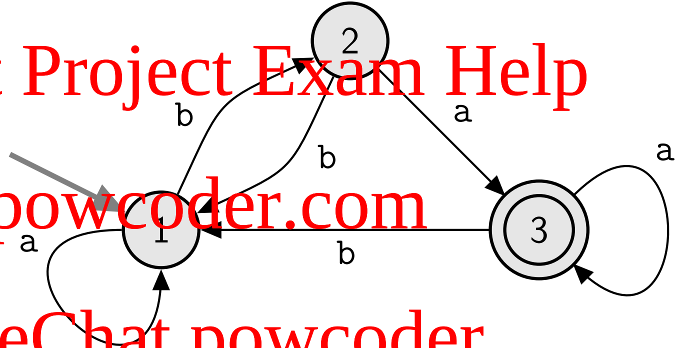
		a	b
Start	{1,2,8}	{2,8}	{3,4,9}
	{2,8}	{2,8}	{3,4,9}
	{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final	{5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final	{6,7,11,12}	{6,7,11,12}	{2,8}

DFA

		a	b
Start	{1,2,8}	{2,8}	{3,4,9}
	{2,8}	{2,8}	{3,4,9}
	{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final	{5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final	{6,7,11,12}	{6,7,11,12}	{2,8}

DFA

		a	b
Start	1	1	2
	2	3	1
Final	3	3	1



Minimum DFA

Algorithm: FA simplification

Input: a FA

Colour all Final States with one colour, and all non-Final States with a different colour.

repeat

  for *each colour used so far* **do**

    Consider all states with that colour.

    if *their rows do not have the same pattern of colours* **then**

      */** States with different colour patterns along their rows must get different colours. So . . . **/*

      Give each different row pattern a different colour, using new colours.

      */** Each set of states having the same row pattern gets the colour for that row pattern. **/*

until *no new colour has been added in this iteration*;

Give each colour a unique number, and use these numbers to form the transition table.

Output: an equivalent FA with fewest states

Other Algorithms

There are algorithms that can take a regular expression and produce a minimum state DFA without constructing a NFA.

There are algorithms that produce fast and more compact representations of a DFA transition table than the straightforward two-dimensional table.

Programming Finite Automata

Once we have a Finite Automaton for a regular expression,
we can write a program for it.

Algorithm: detecting strings accepted by an FA given by a table.

Input: a string

currentState := 1;

table := table with rows (1, 2), (3, 1), (3, 1); $\begin{pmatrix} 1, 2 \\ 3, 1 \\ 3, 1 \end{pmatrix}$

nextLetter := next character of input;

while nextLetter exists **do**

  switch nextLetter **do**

    case 'a' **do**

      currentState := table[currentState][0];

      break;

    case 'b' **do**

      currentState := table[currentState][1];

      break;

  nextLetter := next character of input;

if currentState == 3 **then** print "Match"; **else** print "No Match";

Application: Lexical Analysis

Many situations require text to be divided into substrings that match various *patterns*.

Calculator: `exp(sqrt(-1)*3.14159265)+1`

Programming languages:

```
read(n); sum := 0; i := 1; while(i <= n) {sum += 1.0/i; i++} write(sum).
```


Personnel records:


```
//Employer: Harvard College Observatory.// Annie Jump Cannon,
11/12/1863. Williamina Paton Stevens Fleming, 15/5/1857. Henrietta
Swan Leavitt 4/7/1868. Edward Charles Pickering, 19/7/1846.
```

Application: Lexical Analysis

Many situations require text to be divided into substrings that match various *patterns*.

Calculator:

```
exp ( sqrt ( -1 ) * 3.14159265 ) + 1
```


Programming languages:

```
read ( n ) ; sum := 0 ; i := 1 ;
while ( i <= n ) { sum += 1.0 / i ; i ++ } write ( sum ) .
```


Personnel records:


```
//Employer: Harvard College Observatory.// Annie Jump Cannon ,
11/12/1863 . Williamina Paton Stevens Fleming , 15/5/1857 .
Henrietta Swan Leavitt , 4/7/1868 . Edward Charles Pickering ,
19/7/1846 .
```

Terminology

A **pattern** is specified by a regular expression.

A **token** is a name of a pattern.

- It may also have an attribute value associated with it.

A **lexeme** is a sequence of characters that matches the pattern corresponding to a token.

So a pattern describes the form that the lexemes of a token may take.

Lexical Analyzer

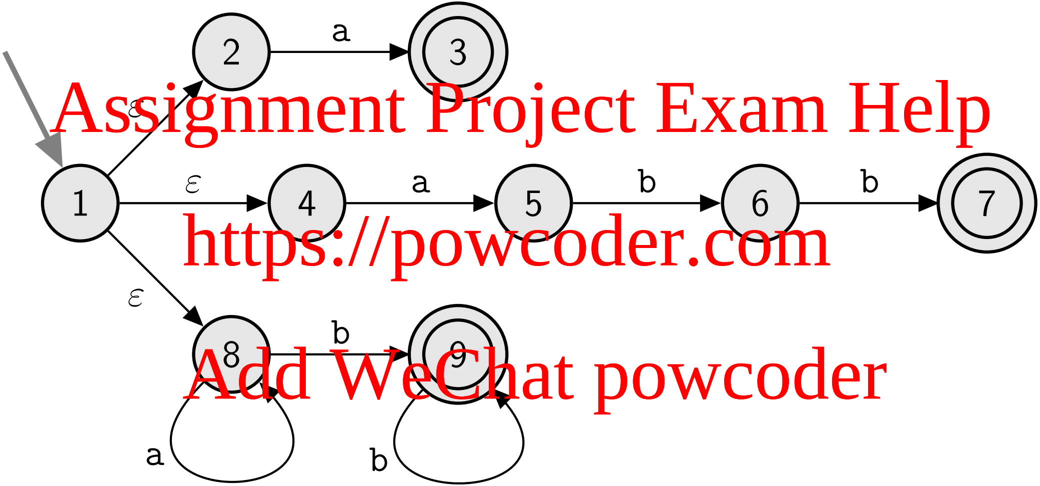
A **lexical analyzer**:

- reads the input one character at a time, and
- splits the input up into lexemes with their associated tokens,
- where each token corresponds to a specific regular language.
- It outputs a sequence of tokens
  (along with any attribute values that any tokens have).
- It is implemented using a Finite Automaton.

Matching Regular Expressions

Write a program which reads in a character string, over alphabet {a,b}, one character at a time and identifies whether or not the string matches one the following regular expressions, and which one.

$$\texttt{a}, \quad \texttt{abb}, \quad \texttt{a}^*\texttt{bb}^*$$


1
2
3
4
5
6
7
8
9
a
a
b
b
b
a
b
ε
ε
ε

Convert to FA, giving:

		a	b
Start	{1,2,4,8}	{3,5,8}	{9}
Final a	{3,5,8}	{8}	{6,9}
Final a*bb*	{9}	∅	{9}
	{8}	{8}	{9}
Final a*bb*	{6,9}	∅	{7,9}
Final abb	{7,9}	∅	{9}
	∅	∅	∅

Conventions

Often it is possible to split a sequence of characters up into tokens in more than one way.

- Consider `abbbb`
- **Convention:** Match the largest possible lexeme at each stage.

Often a sequence of characters can match more than one token.

- Consider `abb`
- **Convention:** If the lexemes are the same length, choose the first token that is listed.

Revision

- Know how to find the DFA with the minimum number of states
- Know how to implement a finite automaton
- Understand what a lexical analyzer does.