Regular Languages

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Objectives

You should be able to...

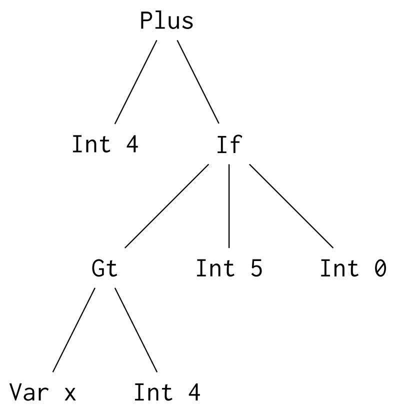
- Be able to explain the problem of parsing.
- Know how to recognize a word using an NFA or a DFA.
- Know the difference between a DFA and an NFA
- Be able to convert a NFA into a DFA
- Vocabulary to know: deterministic, nondeterministic, lexing, scanning, accept state, transition.
- Know the syntax of regular expressions.
- Know how to convert between regular expressions and state machines.
- Know the limitations of regular languages.

The Problem

- Computer programs are entered as a stream of ASCII (usually) characters.

```
4 + if x > 4 then 5 else 0
```

- We want to convert them into an *Abstract Syntax Tree*



Haskell Version

```
PlusExp (IntExp 4)
  (IfExp (GtExp (VarExp "x") (IntExp 4))
        (IntExp 5)
        (IntExp 0))
```

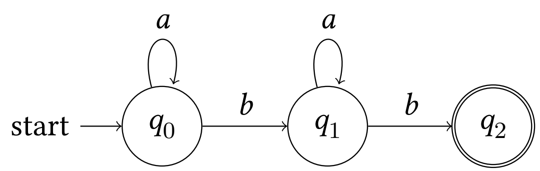
The Solution

- Start with *characters* and convert them into *words*.
  - (a.k.a. *tokens*)
  - This is called *lexing*, *scanning*, or *tokenizing*.
- Convert the tokens into a *tree*.
  - This is called *parsing*.

State Machines

Suppose I want to teach the computer how to recognize a word with the following properties:

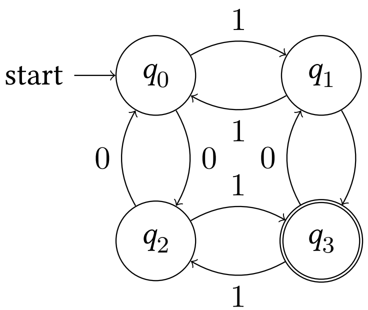
- It consists only of the letters a and b.
- The letter b occurs twice, once at the very end.
- We can use a *state machine*...



- q_0 is the *start state*.
- The transitions consume a character of input.
- q_2 is an *accepting state*.
  - You can have more than one accepting state.
  - You can have transitions out of an accepting state.

Example 1

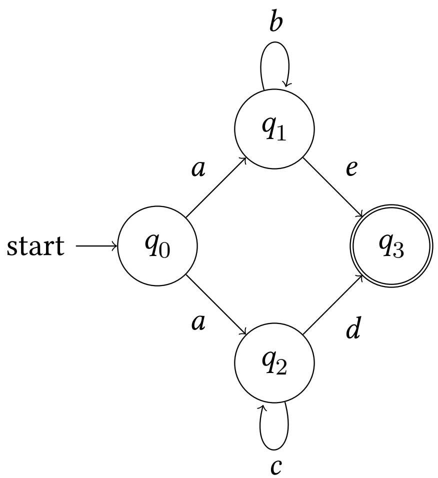
- What kind of strings will this state machine accept?



Nondeterminism

State machines can be *nondeterministic* in two ways:

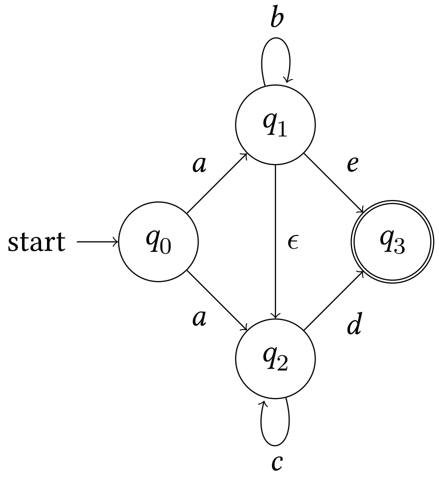
- Way 1: Multiple edges from a state with the same label



Nondeterminism

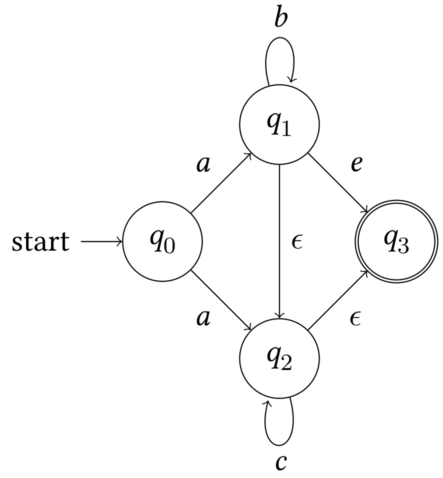
State machines can be *nondeterministic* in two ways:

- Way 2: Have edges that don't consume input.



ϵ-closure

- The ϵ-closure is the set of all states that can be reached by only taking ϵ paths.



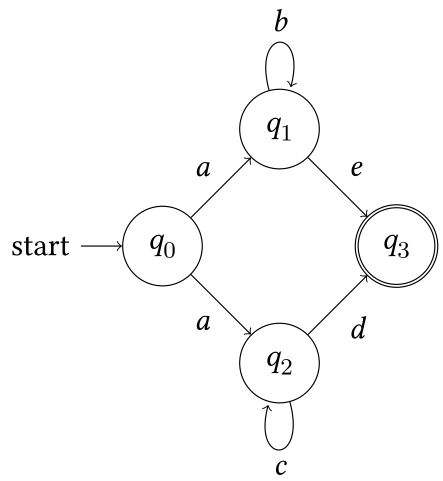
Using State Machines

- With the exception of Prolog, computers have a hard time dealing with nondeterministic state machines.
- Solution: we can convert them!

How to do it:

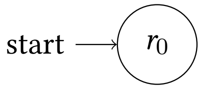
1. Add set $\{q_0\}$ to the queue.
2. Pop set of states Q from the queue. If seen before, discard and go to 1.
3. Take the *epsilon closure* of Q to get R.
4. Create a new state named after R. For each input recognized by q_R, push the resulting set of states onto the queue.

Example 2



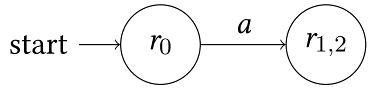
- Start with q_0. Create a new state r_0 in the new machine.

Example 2



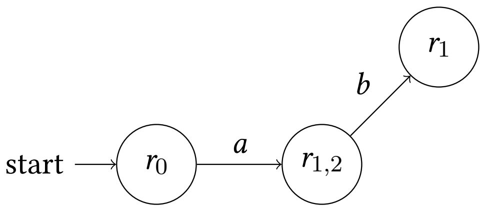
- An a from set q_0 will go to q_1 and q_2. So create a state $r_{1,2}$.

Example 2



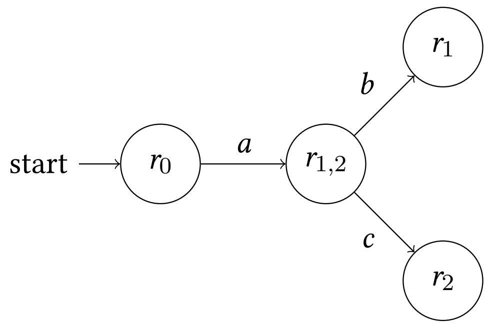
- A b from set q_1, q_2 will go to q_1. So create a state r_1.

Example 2



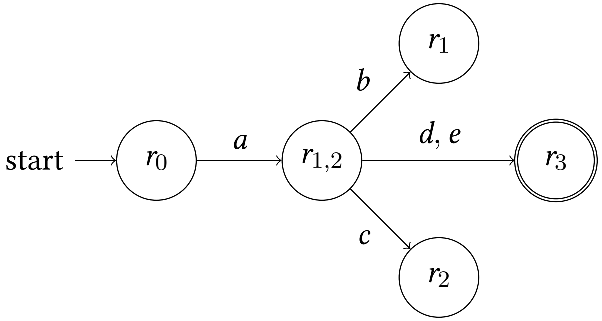
- A c from set q_1, q_2 will go to q_2. So create a state r_2.

Example 2



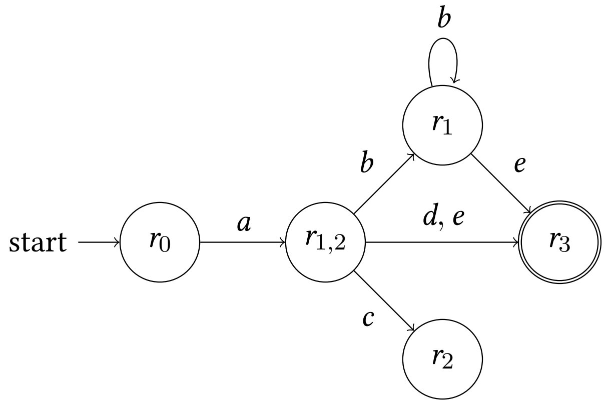
- An e or d from set q_1, q_2 will go to q_3. So create a state r_3.

Example 2



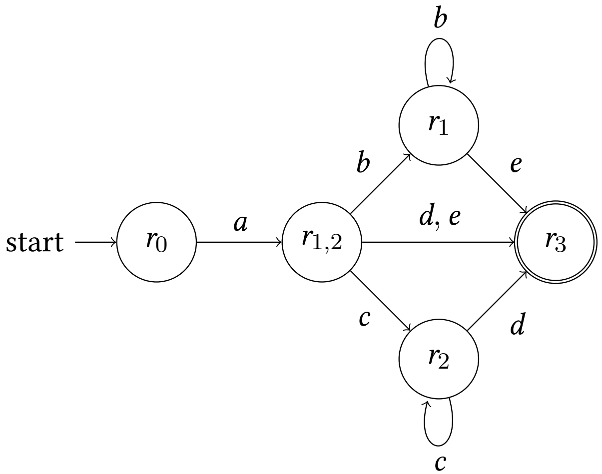
- An e from set q_1 will go to q_3.
- A b from set q_1 will go to q_1.

Example 2

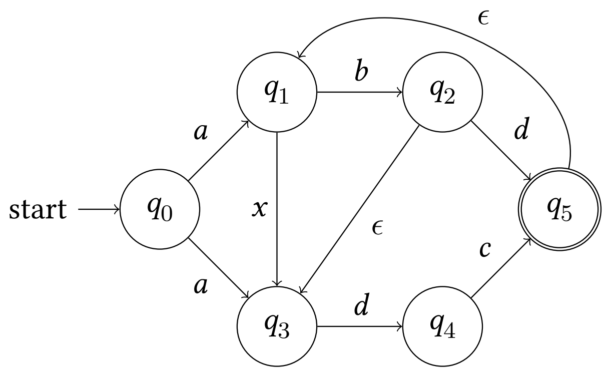


- A d from set q_2 will go to q_3.

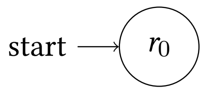
Example 2



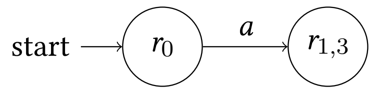
Example 3



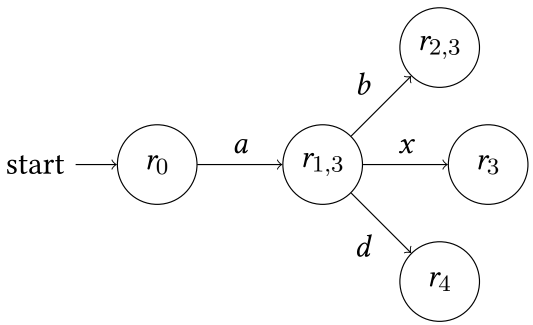
Example 3



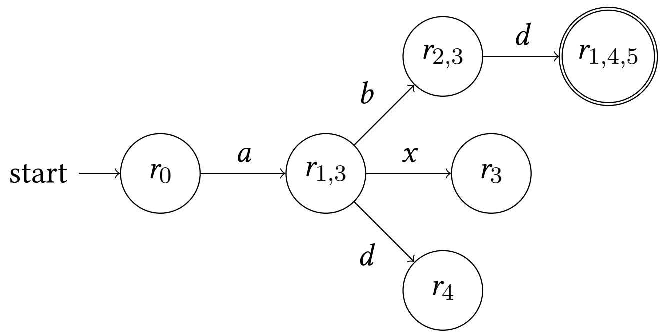
Example 3



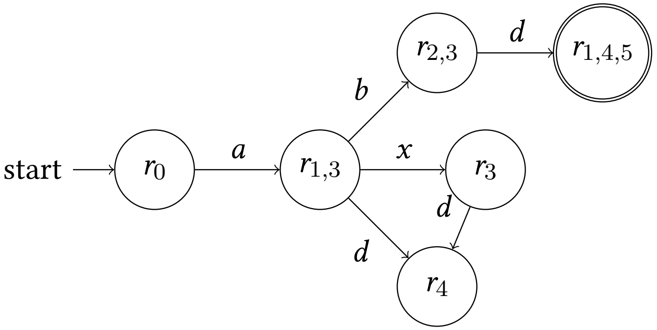
Example 3



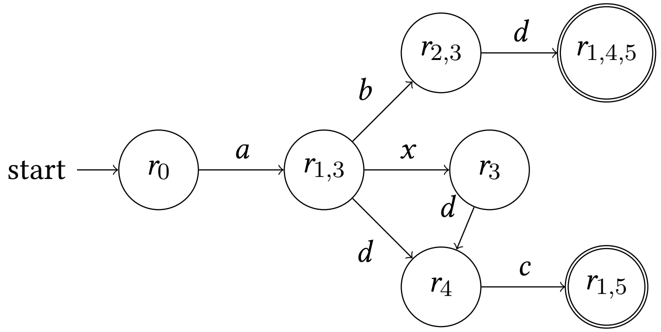
Example 3



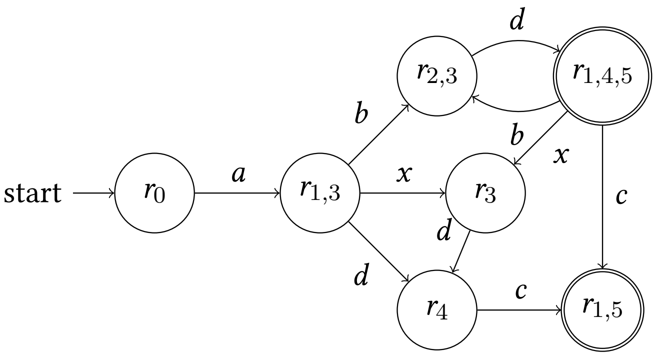
Example 3



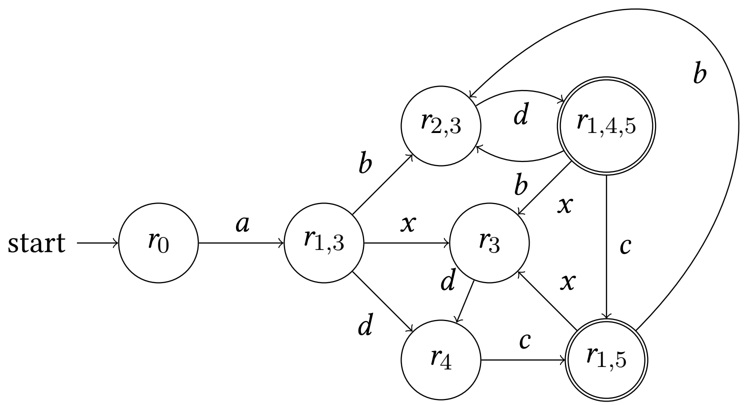
Example 3



Example 3

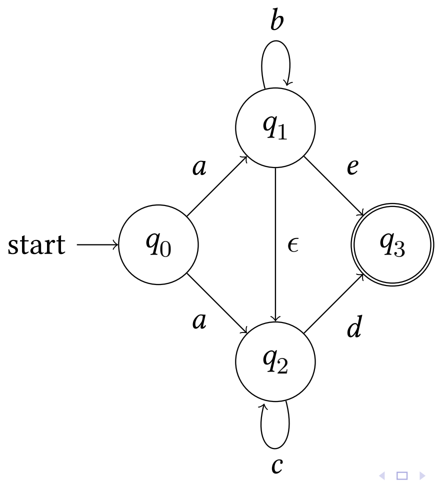


Example 3



Activity!

1. Draw an NFA that accepts even binary numbers.
2. Draw a DFA that accepts even binary numbers.
3. Convert this example into a DFA.



Motivation

- *Regular Languages* were developed by Noam Chomsky in his quest to describe human languages.
- Computer Scientists like them because they are able to describe "words" or "tokens" very easily.

Examples:

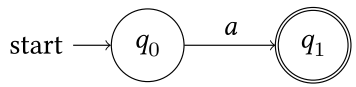
Integers a bunch of digits
Reals an integer, a dot, and an integer
Past Tense English Verbs a bunch of letters ending with "ed"
Proper Nouns a bunch of letters, the first of which must be capitalized

A bunch of digits?!

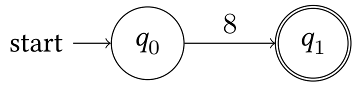
- We need something a bit more formal if we want to communicate properly.
- We will use a *pattern* (or a *regular expression*) to represent the kinds of words we want to describe.
- As it will turn out, these expressions will correspond to NFAs.
- Kinds of patterns we will use:
  - Single letters
  - Repetition
  - Grouping
  - Choices

Single Letters

- To match a single character, just write the character.
- To match the letter "a"...
  - Regular Expression: `a`
  - State machine:

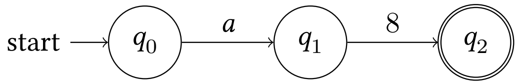


- To match the character "8"...
  - Regular Expression: `8`
  - State machine:

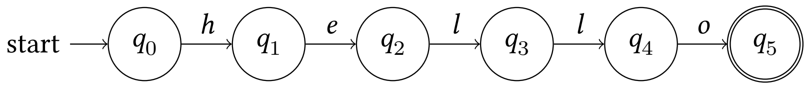


Juxtaposition

- To match longer things, just put two regular expressions together.
- To match the character "a" followed by the character "8"...
  - Regular expression: `a8`
  - State machine:



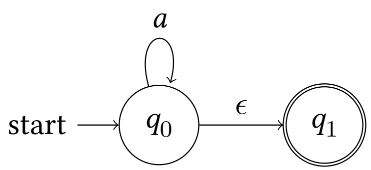
- To match the string "hello"...
  - Regular expression: `hello`
  - State machine:



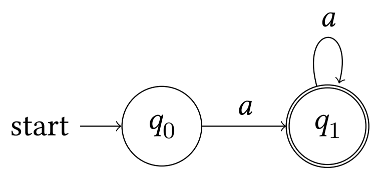
Repetition

- For zero or more occurrences, add a `*`
- For one or more occurrences, add a `+`

- Zero or more copies of `a`...
  - Regular expression `a*`
  - State machine:

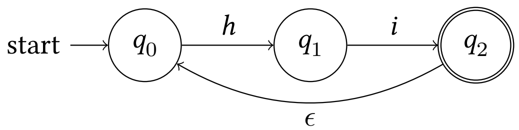


- One or more copies of `a`...
  - Regular expression `a+`
  - State machine:



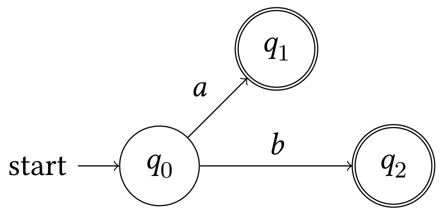
Grouping

- To groups things together, use parenthesis.
- To match one or more copies of the word "hi"...
  - Regular expression: `(hi)+`
  - State machine:



Choice

- To make a choice, use the vertical bar (also called "pipe").
- To match an `a` or a `b`...
  - Regular expression: `a|b`
  - State machine:



Examples

<table>
<tr><th>Expression</th><th>(Some) Matches</th><th>(Some) Rejects</th></tr>
<tr><td>ab*a</td><td>aa, aba, abbba</td><td>ba, aaba, abaa</td></tr>
<tr><td>(0|1)*</td><td>any binary number, ϵ</td><td></td></tr>
<tr><td>(0|1)+</td><td>any binary number</td><td>empty string</td></tr>
<tr><td>(0|1)*0</td><td>even binary numbers</td><td></td></tr>
<tr><td>(aa)*a</td><td>odd number of as</td><td></td></tr>
<tr><td>(aa)*a(aa)*</td><td>odd number of as</td><td></td></tr>
<tr><td>(aa|bb)*((ab|ba)(aa|bb)*(ab|ba)(aa|bb)*)*</td><td>even number of as and b</td><td></td></tr>
</table>

Some Notational Shortcuts

- A range of characters: [Xa-z] matches X and between a and z (inclusively).
- Any character at all: .
- Escape: \

Expression	(Some) Matches
[0-9]+	integers
X.*Y	anything at all between an X and a Y
[0-9]*\.[0-9]*	floating point numbers (positive, without exponents)

Things to know...

- They are *greedy*.
  X.*Y will match XabaaYaababY entirely, not just XabaaY.
- They *cannot count* very well.
  - They can only count as high as you have states in the machine.
  - This regular expression matches some primes:

    aa|aaa|aaaaa|aaaaaaa
  - You cannot match an infinite number of primes.
  - You cannot match “nested comments”. (*.**)

Problems I

Write a regular expression for the following kinds of words

- hexadecimal numbers
- numbers in scientific notation
- file names ending in .c
- numbers between 0 and 255

Describe in English the following regular expressions

- [a-zA-Z][a-zA-Z0-9]+
- [a-z]*(es|ed|ing)
- <[a-z0-9]+@[a-z0-9]+(\.[a-z0-9]+)+>

Answers I

- hexadecimal numbers: `[0-9A-Fa-f]+`
- numbers in scientific notation: `[0-9]+\.[0-9]+E(+|-)[0-9]+`
- file names ending in `.C`: `.*\.C`
- numbers between 0 and 255:
  `25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9]`
- `[a-zA-Z][a-zA-Z0-9]+` like variable names
- `[a-z]*(es|ed|ing)` words ending in “es”, “ed”, or “ing” (verb forms)
- `<[a-z0-9]+@[a-z0-9]+(\.[a-z0-9]+)+>` email addresses

Problems II

Which of the following can be described by regular expressions?

- All the words in the English language
- All the Fibonacci numbers
- “All Your Base Are Belong To Us” video
- Numbers that are multiples of 4 (assume >= 2 digits)
- Words that have exactly as many `as` as they have `bs`
- Palindromes

Answers II

- All the words in the English language
  Yes — it’s huge, but it works. `(a|aardvark|abate|...`
- All the Fibonacci numbers
  No — the set is infinite and requires computation
- “All Your Base Are Belong To Us” video
  Yes — again, huge, but it works
- Numbers that are multiples of 4 (assume >= 2 digits)
  Yes — `[0-9]*([02468][048]|[13579][26])`
- Words that have exactly as many `as` as they have `bs`
  No — requires unbounded counting
- Palindromes
  No — requires unbounded memory
  (aibohphobia = fear of palindromes)