Natural Language Processing
CSCI 4152/6509 — Lecture 4
NFA, Regular Expressions Review, Perl

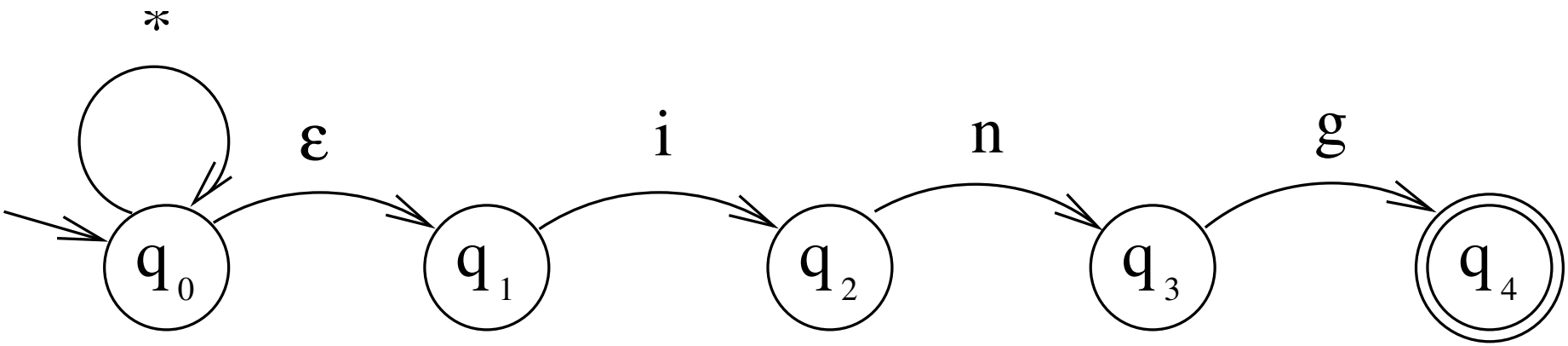
Instructors: Vlado Keselj
Time and date: 16:05 – 17:25, 14-Sep-2023
Location: Rowe 1011

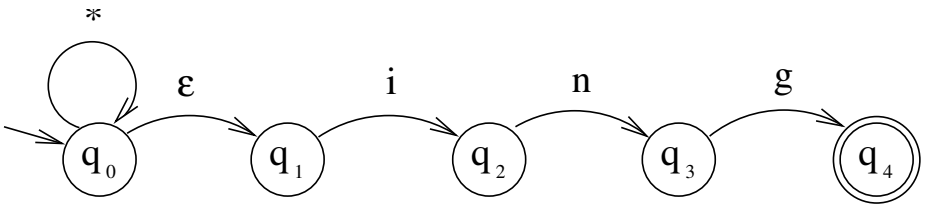
Previous Lecture

- **Part II: Stream-based Text Processing**
- Finite state automata
  - Deterministic Finite Automaton (DFA)
  - Non-deterministic Finite Automaton (NFA)
- Review of Deterministic Finite Automata (DFA)
- Non-deterministic Finite Automata (NFA)
- Implementing NFA, NFA-to-DFA translation (started)

NFA to DFA Example

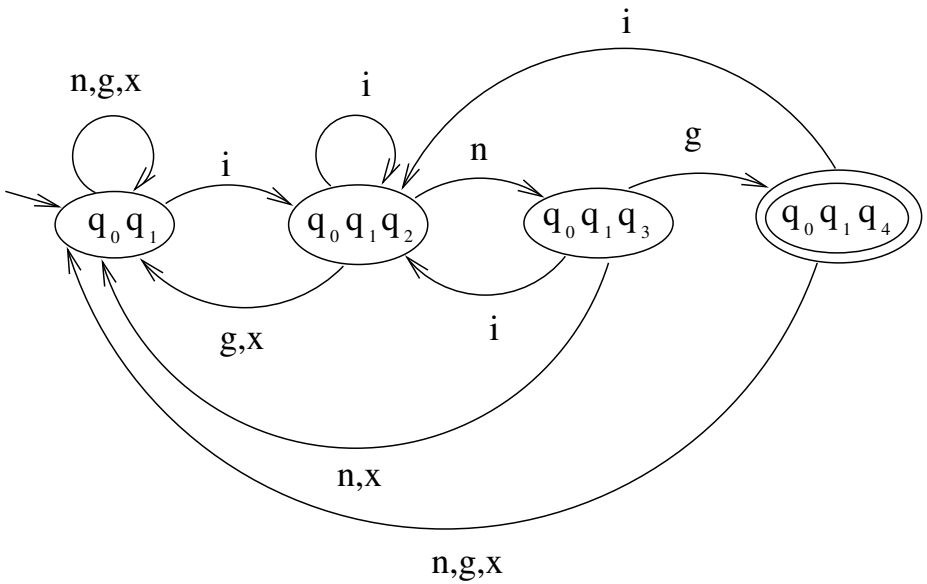
- Let us go back to the example done previously:




*
ε
i
n
g
q0
q1
q2
q3
q4

Final DFA

State	i	n	g	other letters) (not i, n, or g)
$\rightarrow \{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_1\}$
F: $\{q_0, q_1, q_4\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$



Finite Automata in NLP

- Useful in data preprocessing, cleaning, transformation and similar low-level operations on text
- Useful in preprocessing and data preparation
- Efficient and easy to implement
- Regular Expressions are equivalent to automata
- Used in Morphology, Named Entity Recognition, and some other NLP sub-areas

Regular Expressions

- Review (should have been covered in earlier courses as well)
- Used as patterns to match parts of text
- Equivalent to automata, although this may not be obvious
- Provide a compact, algebraic-like way of writing patterns
- Example: `/Submit (the )?file [A-Za-z.-]+/`

Some References on Regular Expressions

You can find many references on Regular Expressions, including:

- Chapter 2 of the textbook [JM]
- Perl "Camel book" or many resources on Internet
- On timberlea server: '`man perlre`' and '`man perlretut`'
- The same effect: '`perldoc perlre`' and '`perldoc perlretut`'
- Or on the web: `http://perldoc.perl.org/perlre.html` and `http://perldoc.perl.org/perlretut.html`

A Historical View on Regular Expressions

- Research by Stephen Kleene: regular sets, and the name of regular sets and regular expressions (1951),
- Implementation in QED by Ken Thompson (1968),
- Open-source implementation by Henry Spencer (1986),
- Use in Perl by Larry Wall (1987),
- Perl-style Regular Expressions in many modern programming languages.

Example Regular Expressions

- Literal: `/woodchuck/` `/Buttercup/`
- Character class: `/./` (any character),
  `/[wW]oodchuck/`, `/[abc]/`, `/[12345]/`
(any of the characters)
- Range of characters: `/[0-9]/`, `/[3-7]/`, `/[a-z]/`,
`/[A-Za-z0-9_-]/`
- Excluded characters and repetition: `/[^()]+/`
- Grouping and disjunction: `/(Jan|Feb) \d?\d/`
- Note: `\d` is same as `[0-9]`
- Another character class: `\w` is same as `[0-9A-Za-z_]`
('word' characters)
- Opposite: `\W` same as `[^0-9A-Za-z_]`

RegEx Examples: Anchors, Grouping, Iteration

```
/^This is a/   # use of anchor
/This^or^that/ # not an anchor
/woodchucks?/
/\bcolou?r\b/        # anchor \b
/is a sentence\.$/  # end of string anchor

# Grouping and iteration:
/This sentence goes on(, and on)*\.$/
/cat|dog/            # disjunction (alternation)
/The (cat|dog) ate the food\./
```

Introduction to Perl

- Created in 1987 by Larry Wall
- Interpreted, but relatively efficient
- Convenient for string processing, system admin, CGIs, etc.
- Convenient use of Regular Expressions
- Larry Wall: Natural Language Principles in Perl
- Perl is introduced in lab in more details

Perl: Some Language Features

- interpreted language, with just-in-time semi-compilation
- dynamic language with memory management
- provides effective string manipulation, brief if needed
- convenient for system tasks
- syntax (and semantics) similar to:
  C, shell scripts, awk, sed, even Lisp, C++

Some Perl Strengths

- **Prototyping:** good prototyping language, expressive: It can express a lot in a few lines of code.
- **Incremental:** useful even if you learn a small part of it. It becomes more useful when you know more; i.e., its learning curve is not steep.
- **Flexible:** e.g, most tasks can be done in more than one way
- **Managed memory:** garbage collection and memory management
- **Open-source:** free, open-source; portable, extensible
- **RegEx support:** powerful, string and data manipulation, regular expressions
- **Efficient:** relatively, especially considering it is an interpreted language
- **OOP:** supports Object-Oriented style

Some Perl Weaknesses

- not as efficient as C/C++
- may not be very readable without prior knowledge
- OO features are an add-on, rather than built-in
- competing popular languages
- not a steep learning curve, but a long one (which is not necessarily a weakness)

Perl in This Course

- Examples in lectures, but you are expected to learn used features by yourself
- Labs will cover more details
- Finding help and reading:
  - Web: `perl.com`, `CPAN.org`, `perlmonks.org`, ...
  - `man perl`, `man perlintro`, ...
  - books: e.g., the "Camel" book: "Learning Perl, 4th Edition" by Brian D. Foy; Tom Phoenix; Randal L. Schwartz (2005)

Testing Code

- Login to timberlea
- Use plain editor, e.g., emacs
- Develop and test program
- Submit assignments
- You can use your own computer, but code must run on timberlea

Perl File Names

- Extension '`.pl`' is common, but not mandatory
- `.pl` is used for programs (scripts) and basic libraries
- Extension '`.pm`' is used for Perl modules

"Hello World" Program

Choose your favorite editor and edit `hello.pl`:

```
print "Hello world!\n";
```

Type "`perl hello.pl`" to run the program, which should produce: `Hello world!`

Another way to run a program

Let us edit again `hello.pl` into:

```
#!/usr/bin/perl
print "Hello world!\n";
```

Change permissions of the program and run it:

```
chmod u+x hello.pl
./hello.pl
```

Simple Arithmetic

```
#!/usr/bin/perl
print 2+3, "\n";
$x = 7;
print $x * $x,"\n";
print "x = $x\n";
```

Output:

```
5
49
x = 7
```

Direct Interaction with Interpreter

- Command: `perl -d -e 1`
- Enter commands and see them executed
- 'q' to exit
- This interaction is through Perl debugger

Syntactic Elements

- statements separated by semi-colon ';'
- white space does not matter except in strings
- line comments begin with '#'; e.g.
  `# a comment until the end of line`
- variable names start with $, @, or % ('sigils'):
  $a — a scalar variable
  @a — an array variable
  %a — an associative array (or hash)
  However: `$a[5]` is 5th element of an array @a, and
  `$a{5}` is a value associated with key 5 in hash %a
- the starting special symbol is followed either by a name (e.g., `$varname`) or a non-letter symbol (e.g., `$!`)
- user-defined subroutines are usually prefixed with &:
  &a — call the subroutine a (procedure, function)

Example Program: Reading a Line

```
#!/usr/bin/perl
use warnings;

print "What is your name?  ";
$name = <>; # reading one line of input
chomp $name; # removing trailing newline
print "Hello $name!\n";
```

`use warnings;` enables warnings — recommended!
`chomp` removes the trailing newline from `$name` if there is one. However, changing the special variable `$/` will change the behaviour of `chomp` too.

Example: Declaring Variables

The declaration "`use strict;`" is useful to force more strict verification of the code. If it is used in the previous program, Perl will complain about variable `$name` not being declared, so you can declare it: `my $name`

We can call this program `example3.pl`:

```
#!/usr/bin/perl
use warnings;
use strict;

my $name;
print "What is your name?  ";
$name = <>;
chomp $name;
print "Hello $name!\n";
```

Perl Program for Counting Lines

```
#!/usr/bin/perl
# program:  lines-count.pl

while (<>) {
   ++$count;
}

print "$count\n";
```