Natural Language Processing
CSCI 4152/6509 — Lecture 3
Finite Automata Review

Instructors: Vlado Keselj
Time and date: 14:35 – 15:55, 2-Oct-2025
Location: Studley LSC-Psychology P5260

Previous Lecture

- Why is NLP hard?
  - ambiguous, vague, universal
- Ambiguities at different levels of NLP
- About course project
  - Deliverables: P0, P1, P, R
  - Project report structure
  - Choosing project topic
- **Part II: Stream-based Text Processing**
- Finite state automata (start)

Consider DFA for: `ha-ha-...-ha`

Representing DFA

- Formally, as sets and functions (mappings)
- As a transition table
- As a graph
- Consider the DFA for the language:
  `baaa...a!`

DFA for language `baa...a!` using a table

Non-deterministic Finite Automaton

- Formally: $(Q, \Sigma, \delta, q_0, F)$
- However, the transition function is different:
  $\delta : Q \times \Sigma_\varepsilon \to P(Q)$
  where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, and $P(Q)$ is the set of all subsets of Q (powerset)
- A string is accepted if there is *at least* one path leading to an accepting state
- Consider: `/.*ing/` or `/jan|jun|jul/`

NFA for `/.*ing/` or `/jan|jun|jul/`

Another NFA and DFA Example

- Write a DFA that accepts any sequence over alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$ that ends with '`eses`', like 'theses' or 'parentheses'.
- Write an NFA that accepts the same language.

Implementing NFAs

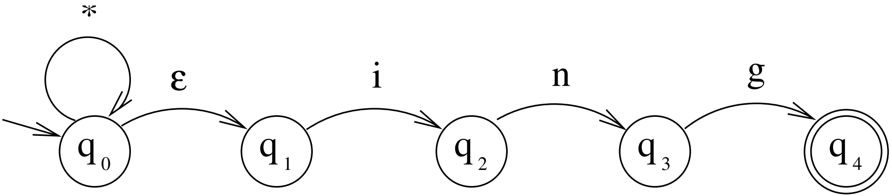
- DFA — easy to implement, NFA — not straightforward
- Two approaches for NFA: backtracking and translation to DFA
- Using backtracking — usually inefficient solution
- Translating into a DFA
  - Sets of reachable NFA states become states of new DFA

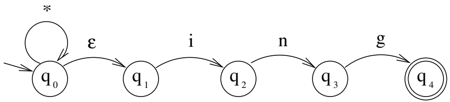
NFA to DFA Translation

- Start with NFA and create new equivalent DFA
- DFA states are sets of NFA states
- If q_0 is the start NFA state, then the start DFA state is **Closure**(q_0)
- **Closure**(A) of a set of NFA states A is a set A with all states reachable via ε-transitions from A
- Fill DFA transition table by keeping track of all states reachable after reading next input character
- Final states in DFA are all sets that contain at least one final state from NFA

NFA to DFA Example

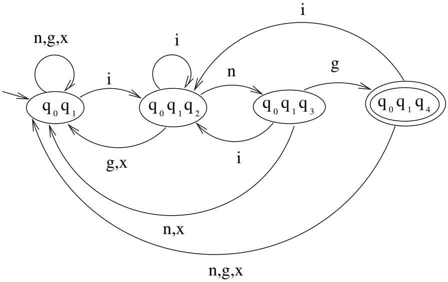
- Let us go back to the example done previously:

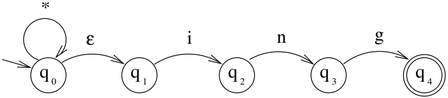



*
ε
i
n
g
q0
q1
q2
q3
q4

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: $\{q_0, q_1, q_4\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$





(to finish)

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 of regular expressions (for some of you, it was 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\./
```