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

Instructors: Vlado Keselj
Time and date: 16:05 – 17:25, 11-Sep-2024
Location: Carleton Tupper Building Theatre C

Previous Lecture

- 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

- Considering text as a stream of characters, words, and lines of text
- Review of Finite Automata and Regular Expressions
- Review of Unix-style text processing
- Introduction to Perl
- Morphology fundamentals
- N-grams
- Reading: Chapter 2, Jurafsky and Martin

Finite-State Automata

- Regular Expressions and Regular Languages
- Regular Languages can be described using
  - Regular Expressions
  - Regular Grammars
  - Finite-State Automata (DFA and NFA)
- DFA = Deterministic Finite Automaton
- NFA = Non-deterministic Finite Automaton
- also referred to as Finite-State Machines

Typical Low-level NLP Tasks

- Pre-processing text
- Tokenization
- Sentence Segmentation
- Morphological Processing (e.g., Stemming)
- "Vectorizing" Text
- Information Extraction (simpler cases)
- *and so on*

Example Task: Removing HTML Tags

Deterministic Finite Automaton

- Formally defined as a 5-tuple: $(Q, \Sigma, \delta, q_0, F)$
  - Q is a set of states
  - Σ is an input alphabet
  - $\delta : Q \times \Sigma \to Q$ is a transition function
  - $q_0 \in Q$ is the start state
  - $F \subset Q$ is a set of final or accepting states
- Graph representation is frequently used
- Consider finite automata for sets of strings:
  `baaa...a!` `ha-ha-...-ha`
  `up-up-down-up-down-up-up-...down`

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

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:
  $\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$
  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 $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$ that ends with '`eses`', like 'theses' or 'parentheses'.
- Write an NFA that accepts the same language.

Implementing NFAs

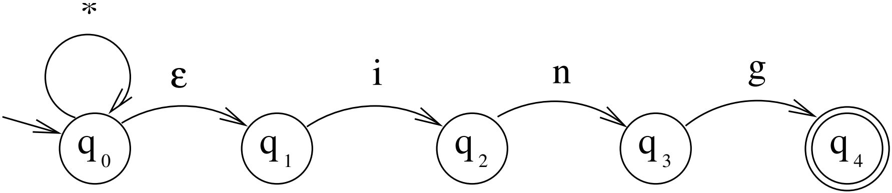
- DFA — easy to implement, NFA — not straightforward
- Two approaches for NFA: backtracking and translation to DFA
- Using backtracking — usually inefficient solution
- Translating into a DFA
  - Sets of reachable NFA states become states of new DFA

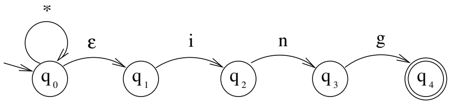
NFA to DFA Translation

- 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
q_0
q_1
q_2
q_3
q_4

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