Overview

Finite-state technology is:

- Fast and efficient
- Useful for a variety of language tasks

Three main topics we'll discuss:

- Regular Expressions (REs)
- Finite-State Automata (FSAs)
- Properties of Regular Languages

REs and FSAs are mathematically equivalent, but help us approach problems in different ways.

Some useful tasks involving language

- Find all phone numbers in a text, e.g., occurrences such as
  When you call (614) 292-8833, you reach the fax machine.
- Find multiple adjacent occurrences of the same word in a text, as in
  I read the the book.
- Determine the language of the following utterance: French or Polish?
  Czy pasazer jadacy do Warszawy moze jechac przez Londyn?

More useful tasks involving language

- Look up the following words in a dictionary:
  laughs, became, unidentifiable, Thatcherization
- Determine the part-of-speech of words like the following, even if you can't find them in the dictionary:
  conurbation, cadence, disproportionality, lyricism, parlance

⇒ Such tasks can be addressed using so-called finite-state machines.
⇒ How can such machines be specified?

Regular expressions

- A regular expression is a description of a set of strings, i.e., a language.
- They can be used to search for occurrences of these strings
- A variety of unix tools (grep, sed), editors (emacs), and programming languages (perl, python) incorporate regular expressions.
- Just like any other formalism, regular expressions as such have no linguistic contents, but they can be used to refer to linguistic units.

The syntax of regular expressions (1)

Regular expressions consist of

- strings of characters: \(c, A100, \text{natural language}, 30\ \text{years!}\)
- disjunction:
  - ordinary disjunction: \(\text{devoured|ate, family|ies}\)
  - character classes: \([Tt]he, bec[oa]me\)
  - ranges: \([A-Z]\) (a capital letter)
- negation: \([^a]\) (any symbol but a)
  \([^A-Z0-9]\) (not an uppercase letter or number)
The syntax of regular expressions (2)

- counters
  - optionality: ?
  - any number of occurrences: *(Kleene star)
    - [0-9] years
  - at least one occurrence: +
    - [0-9]+ dollars
- wildcard for any character: .
- Parentheses to group items together
  - ant(farm)?
- Escaped characters to specify characters with special meanings:
  - \*, \+, \?, \(, \), \|, \[, \]

Additional functionality for some RE uses (1)

Although not a part of our discussion about regular languages, some tools (e.g., Perl) allow for more functionality

Anchors: anchor expressions to various parts of the string
- ^ = start of line
- do not confuse with [ˆ... ] used to express negation
- $ = end of line
- \b non-word character
- word characters are digits, underscores, or letters, i.e., [0-9A-Za-z]

Some RE practice

- What does $[0-9]+(\.[0-9][0-9])$ signify?
- Write a RE to capture the times on a digital watch (hours and minutes). Think about:
  - the (im)possible values for the hours
  - the (im)possible values for the minutes

Formal language theory

We will view any formal language as a set of strings

- The language uses a finite vocabulary Σ (called an alphabet), and a set of string-combining operations
- Regular languages are the simplest class of formal languages
  - class of languages definable by REs
  - class of languages characterizable by FSAs
Finite state machines

Finite state machines (or automata) (FSM, FSA) recognize or generate regular languages, exactly those specified by regular expressions.

Example:

- **Regular expression**: `colou?r`
- **Finite state machine (representation):**

Properties of regular languages (1)

The regular languages are closed under \((L_1, L_2 \text{ regular languages})\):

- **concatenation**: \(L_1 \cdot L_2\)
  set of strings with beginning in \(L_1\) and continuation in \(L_2\)
- **Kleene closure**: \(L^*\)
  set of repeated concatenation of a string in \(L\)
- **union**: \(L_1 \cup L_2\)
  set of strings in \(L_1\) or in \(L_2\)
- **complementation**: \(\Sigma^* - L_1\)
  set of all possible strings that are not in \(L_1\)

Properties of regular languages (2)

The regular languages are closed under \((L_1, L_2 \text{ regular languages})\):

- **difference**: \(L_1 - L_2\)
  set of strings which are in \(L_1\) but not in \(L_2\)
- **intersection**: \(L_1 \cap L_2\)
  set of strings in both \(L_1\) and \(L_2\)
- **reversal**: \(L_1^R\)
  set of the reversal of all strings in \(L_1\)

What sorts of expressions aren’t regular?

In natural language, examples include *center-embedding* constructions.

- These dependencies are not regular:
  1. The cat loves Mozart.
  2. The cat the dog chased loves Mozart.
  3. The cat the dog the rat bit chased loves Mozart.
  4. The cat the dog the rat the elephant admired bitten loves Mozart.

- Similar ones would be regular:
  1. A*B* loves Mozart

Accepting/Rejecting strings

The behavior of an FSA is completely determined by its transition table.

- The assumption is that there is a tape, with the input symbols read off consecutive cells of the tape.
  - The machine starts in the start (initial) state, about to read the contents of the first cell on the input tape.
  - The FSA uses the transition table to decide where to go at each step
- A string is rejected in exactly two cases:
  1. a transition on an input symbol takes you nowhere
  2. the state you're in after processing the entire input is not an accept (final) state
- Otherwise, the string is accepted.
Defining finite state automata

A finite state automaton is a quintuple \((Q, \Sigma, E, S, F)\) with

- \(Q\) a finite set of states
- \(\Sigma\) a finite set of symbols, the alphabet
- \(S \subseteq Q\) the set of start states
- \(F \subseteq Q\) the set of final states
- \(E\) a set of edges \(Q \times (\Sigma \cup \{\epsilon\}) \times Q\)

The transition function \(d\) can be defined as
\[
d(q, a) = \{q' \in Q | \exists (q, a, q') \in E\}
\]

Example FSA

FSA to recognize strings of the form: \([ab]^+\)

- \(\Sigma = \{a, b\}\)
- \(Q = \{0, 1\}\)
- \(S = \{0\}\)
- \(F = \{1\}\)
- \(E = \{(0, a, 1), (0, b, 1), (1, a, 1), (1, b, 1)\}\)

FSA: set of zero or more a’s

\(L = \{\epsilon, a, aa, aaa, aaaa, \ldots\}\)

- \(Q = \{0\}\)
- \(\Sigma = \{a\}\)
- \(S = \{0\}\)
- \(F = \{0\}\)
- \(E = \{(0, a, 0)\}\)

FSA: set of all lowercase alphabetic strings ending in b

- \(L = \{a, b, ab, ba, aab, bab, aba, bba, \ldots\}\)

- \(Q = \{0, 1\}\)
- \(\Sigma = \{a, b\}\)
- \(S = \{0\}\)
- \(F = \{1\}\)
- \(E = \{(0, a, 0), (0, b, 1), (1, a, 1), (1, b, 1)\}\)

How would we change this to make it: \(\backslash b[a-z]*b\) \(b\)

FSA: the set of all strings in \([ab]^*\) with exactly 2 a’s

Do this yourself

It might help to first rewrite a more precise regular expression for this

- First, be clear what the domain is (all strings in \([ab]^*\))
- And then figure out how to narrow it down

Language accepted by an FSA

The extended set of edges \(\hat{E} \subseteq Q \times \Sigma^* \times Q\) is the smallest set such that

- \(\forall (q, \sigma, q') \in E: (q, \sigma, q') \in \hat{E}\)
- \(\forall (q_0, \sigma_1, q_1), (q_1, \sigma_2, q_2) \in \hat{E}: (q_0, \sigma_1 \sigma_2, q_2) \in \hat{E}\)

The language \(L(A)\) of a finite state automaton \(A\) is defined as
\[
L(A) = \{w | q_s \in S, q_f \in F, (q_s, w, q_f) \in \hat{E}\}
\]
FSA for simple NPs

Where $d$ is an alias for determiners, $a$ for adjectives, and $n$ for nouns:

- $Q = \{0, 1, 2\}$
- $\Sigma = \{d, a, n\}$
- $S = \{0\}$
- $F = \{2\}$
- $E = \{(0, d, 1), (0, \epsilon, 1), (1, a, 1), (1, n, 2), (2, n, 2)\}$

Finite state transition networks (FSTN)

Finite state transition networks are graphical descriptions of finite state machines:

- nodes represent the states
- start states are marked with a short arrow
- final states are indicated by a double circle
- arcs represent the transitions

Example for a finite state transition network

Regular expression specifying the language generated or accepted by the corresponding FSM: $ab|cb^+$

Finite state transition tables

Finite state transition tables are an alternative, textual way of describing finite state machines:

- the rows represent the states
- start states are marked with a dot after their name
- final states with a colon
- the columns represent the alphabet
- the fields in the table encode the transitions

The example specified as finite state transition table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
<td>S3</td>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>S3:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>S2,S3:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some properties of finite state machines

- Recognition problem can be solved in linear time (independent of the size of the automaton).
- There is an algorithm to transform each automaton into a unique equivalent automaton with the least number of states.
Transducers and determinization

A finite state transducer understood as consuming an input and producing an output cannot generally be determined.

Example:

Finite state transducers

A finite state transducer is a 6-tuple \((Q, \Sigma_1, \Sigma_2, E, S, F)\) with

- \(Q\) a finite set of states
- \(\Sigma_1\) a finite set of symbols, the input alphabet
- \(\Sigma_2\) a finite set of symbols, the output alphabet
- \(S \subseteq Q\) the set of start states
- \(F \subseteq Q\) the set of final states
- \(E\) a set of edges \(Q \times (\Sigma_1 \cup \{\epsilon\}) \times \Sigma_2 \times (\Sigma_2 \cup \{\epsilon\})\)

Summary

- Notations for characterizing regular languages:
  - Regular expressions
  - Finite state transition networks
  - Finite state transition tables
- Finite state machines and regular languages: Definitions and some properties
- Finite state transducers