Regular Expressions and Finite-State Automata

L545
Spring 2013

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`, `natural language`, `30 years!`
- disjunction:
  - ordinary disjunction: `devoured|ate`, `famil(y|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: `?`
    `colou?r`
  - any number of occurrences: `*` (Kleene star)
    `[0-9]* years`
  - at least one occurrence: `+`
    `[0-9]+ dollars`
- wildcard for any character: .
  `beg.n` for any character in between `beg` and `n`
- Parentheses to group items together
  `ant(farm)?`
- Escaped characters to specify characters with special meanings:
  `\*, \+, \?, \(, \), \|, \[, \]`

The syntax of regular expressions (3)

Operator precedence, from highest to lowest:

parentheses `()`

counters `* + ?`

character sequences

disjunction `|`

- `fire|ing` = *fire* or *ing*
- `fir(e|ing)` = *fir* followed by either *e* or *ing*

Note: The various unix tools and languages differ w.r.t. the exact syntax of the regular expressions they allow.

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 `[^...]` used to express negation
- `$` = end of line
- `\b` non-word character
  - word characters are digits, underscores, or letters, i.e., `[0-9A-Za-z_]`

Additional functionality for some RE uses (2)

Use **aliases** to designate particular recurrent sets of characters

- `\d` = `[0-9]`: digit
- `\D` = `[^\d]`: non-digit
- `\w` = `[a-zA-Z0-9_]`: alphanumeric
- `\W` = `[^\w]`: non-alphanumeric
- `\s` = `[\r\t\n\f]`: whitespace character
  – `\r`: space, `\t`: tab, `\n`: newline, `\f`: formfeed
- `\S [^\s]`: non-whitespace

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

Regular languages

How can the class of regular languages which is specified by regular expressions be characterized?

Let Σ be the set of all symbols of the language, the alphabet, then:

1. $\{\}$ is a regular language
2. $\forall a \in \Sigma$: $\{a\}$ is a regular language
3. If L_1 and L_2 are regular languages, so are:
   (a) the concatenation of L_1 and L_2: $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$
   (b) the union of L_1 and L_2: $L_1 \cup L_2$
   (c) the Kleene closure of L: $L^* = L_0 \cup L_1 \cup L_2 \cup$ ... where L_i (roughly) represents L concatenated with itself i times, i.e.,
   $L_0 = \{\epsilon\}$, $L_{i+1} = \{ab | a \in L_i$ and $b \in L\}$

Properties of regular languages (1)

The regular languages are closed under (L_1 and L_2 regular languages):

- concatenation: $L_1 \cdot L_2$
  set of strings with beginning in L_1 and continuation in L_2
- Kleene closure: L_1^*
  set of repeated concatenation of a string in L_1
- 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 and L_2 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) a. The cat loves Mozart.
      b. The cat the dog chased loves Mozart.
      c. The cat the dog the rat bit chased loves Mozart.
      d. The cat the dog the rat the elephant admired bit chased loves Mozart.

  (2) (the noun)^n (transitive-verb)$^{n-1}$ loves Mozart

- Similar ones would be regular:

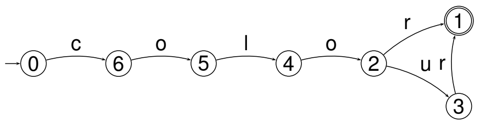
  (3) A*B* loves Mozart

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):



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, Σ, E, S, F) with

- Q a finite set of states
- Σ 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]+`

- i.e., L = { a, b, ab, ba, aab, bab, aba, bba, ...}

FSA is defined as:

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

FSA: set of zero or more a's

L = { ϵ, a, aa, aaa, aaaa, ...}

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

FSA: set of all lowercase alphabetic strings ending in b

L captured by `[a-z]*b`

= {b, ab, tb, ..., aab, abb, ...}

- $Q = \{0, 1\}$
- $\Sigma = \{a, b, c, \ldots, z\}$
- $S = \{0\}$
- $F = \{1\}$
- $E = \{(0, a, 0), (0, c, 0), (0, d, 0), \ldots, (0, z, 0)$
  $(0, b, 1), (1, b, 1),$
  $(1, a, 0), (1, c, 0), (1, d, 0), \ldots (1, z, 0)\}$

How would we change this to make it: `\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

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

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:

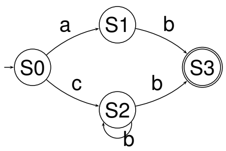
- $Q = \{0, 1, 2\}$
- $\Sigma = \{d, a, n\}$
- $S = \{0\}$
- $F = \{2\}$
- $E = \{(0, d, 1), (0, \epsilon, 1)(1, a, 1), (1, n, 2), (2, n, 2)\}$

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

	a	b	c	d
S0.	S1		S2	
S1		S3:		
S2		S2,S3:		
S3:				

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.

Deterministic Finite State Automata

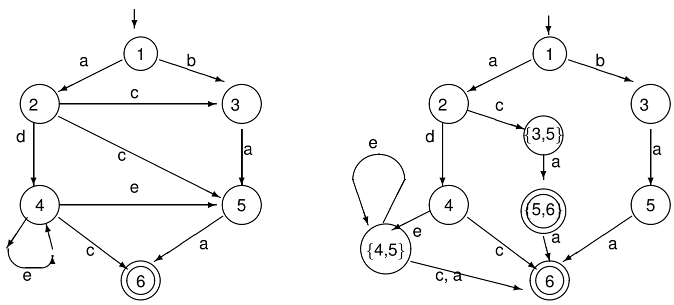
A finite state automaton is deterministic iff it has

- no ϵ transitions and
- for each state and each symbol there is at most one applicable transition.

Every non-deterministic automaton can be transformed into a deterministic one:

- Define new states representing a disjunction of old states for each non-determinacy which arises.
- Define arcs for these states corresponding to each transition which is defined in the non-deterministic automaton for one of the disjuncts in the new state names.

Example: Determinization of FSA



Why finite-state?

Finite number of states

- Number of states bounded in advance – determined by its transition table
- Therefore, the machine has a limit to the amount of memory it uses.
  - Its behavior at each stage is based on the transition table, and depends just on the state its in, and the input.
  - So, the current state reflects the history of the processing so far.

Classes of formal languages which are not regular require additional memory to keep track of previous information, e.g., center-embedding constructions

From Automata to Transducers

Needed: mechanism to keep track of path taken

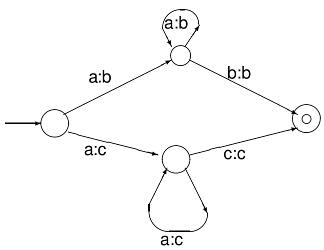
A **finite state transducer** is a 6-tuple $(Q, \Sigma_1, \Sigma_2, E, S, F)$ with

- Q a finite set of states
- Σ_1 a finite set of symbols, the input alphabet
- Σ_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 $Q \times (\Sigma_1 \cup \{\epsilon\}) \times Q \times (\Sigma_2 \cup \{\epsilon\})$

Transducers and determinization

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

Example:



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