Finite-State Machines and Regular Languages

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01

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`

The syntax of regular expressions (3)

Operator precedence, from highest to lowest:

parentheses `( )`

counters `* + ?`

character sequences

disjunction `|`

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

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 is the language of all strings of length i.

Properties of regular languages

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
- difference: $L_1 - L_2$
  set of strings which are in L_1 but not in L_2

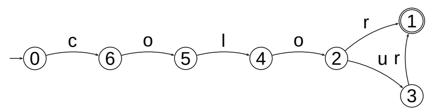
- intersection: $L_1 \cap L_2$
  set of strings in both L_1 and L_2
- reversal: L_1^R
  set of the reversal of all strings in L_1

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:



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

  $d(q, a) = \{q' \in Q | \exists (q, a, q') \in E\}$

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

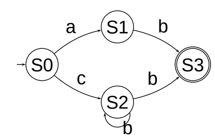
$L(A) = \{w | q_s \in S, q_f \in F, (q_s, w, q_f) \in \hat{E}\}$

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

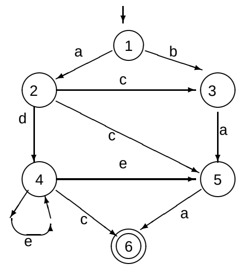
A finite state automaton is deterministic iff it has

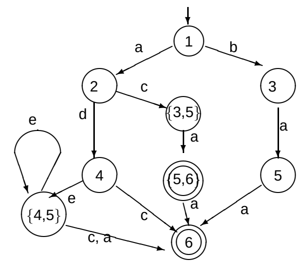
- no ϵ transitions and
- for each state and each symbol there is at most one applicable transition.

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





From Automata to Transducers

Needed: mechanism to keep track of path taken

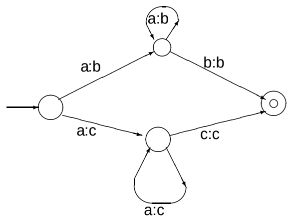
A **finite state transducer** is a 6-tuple $(Q, \Sigma_1, \Sigma_2, E, S, F)$ with

- 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

Reading assignment 2

- Chapter 1 “Finite State Techniques” of course notes
- Chapter 2 “Regular expressions and automata” of Jurafsky and Martin (2000)