Finite state machines and regular languages

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

Regular expressions

A regular expression (RE) is a description of a set of strings, a language.

- can be used to search for occurrences of these strings
- used in a variety of tools: grep, editors, corpus search tools (cqp), . . .
- Just like any other formalism, REs have no linguistic contents as such. But they can well be used to refer to units of morphological or phonological relevance.

Some linguistically informed uses

- Determine the language of the following utterance: French or Polish?

  Czy pasazer jadacy do Warszawy moze jechac przez Londyn?

  ⇒ Knowledge of morphologically/phonologically possible sequences of letters can be used for this task.
- Look up the following words in the dictionary:

  laughs, became, unidentifiable, Thatcherization

  ⇒ Knowledge of morphological composition needed.

Basic regular expressions (1)

Regular expressions consist of

- strings of characters (case sensitive!):
  `c`, `natural language`, `100 years!`
- disjunction:
  - ordinary disjunction: `|`
    `devoured|ate`, `famil(y|ies)`
  - character classes:
    `[Tt]he`, `bec[oa]me`
  - ranges:
    `[A-Z]` for a capital letters
- negation: ˆ as first letter after `[`
  `[ˆa]` any symbol but a
  `[ˆA-Z]` not an uppercase letter

Basic regular expressions (2)

- counters
  - optionality: ?
    `colou?r`
  - any number of occurrences: * (Kleene star)
    `[0-9]* years`
  - at least one occurrence: +
    `$ [0-9]+`
- wildcard for any character: .
  `beg.n` for any character in between `beg` and `n`

Operator precedence, from highest to lowest:

parenthesis `()`

counters `* + ?`

character sequences

disjunction `|`

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:

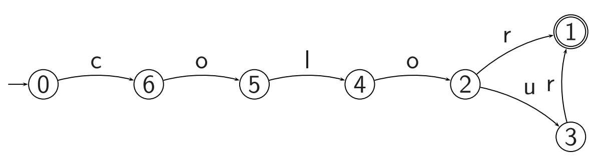
1. $\{\}$ is a regular language
2. $\forall a \in \Sigma$: $\{a\}$ is a regular language
3. If L_1 and L_2 are regular languages, so are:
   (a) the concatenation of L_1 and L_2:
   $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$
   (b) the union (or disjunction) of L_1 and L_2:
   $L_1 \cup L_2$
   (c) the Kleene closure of L_1:
   L_i^*

Finite state machines

Finite state machines (FSM), also called finite state automata (FSA) can recognize or generate regular languages, such as those specified by regular expressions.

Example:

- Regular expression: `colou?r`
- Finite state machine:



Finite state automaton

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 $Q \times (\Sigma \cup \{\epsilon\}) \times Q$

  A **transition function** d can be defined as

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

Language accepted by an FSA

Auxiliary concept: 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

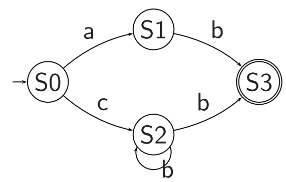
$L(A) = \{w | q_s \in S, q_f \in F, (q_s, w, q_f) \in \hat{E}\}$

Finite state transition networks

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

Simple example:



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

Our simple example:

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

Properties of regular languages

Let L_1 and L_2 be regular languages.

The regular languages are closed under

- 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

Further properties

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

Deterministic Finite State Automata

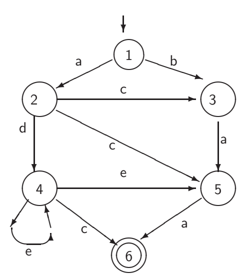
A finite state automaton is deterministic iff it has

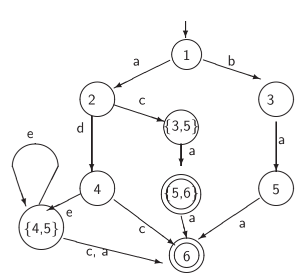
- no ϵ transitions and
- for each state and each symbol there is at most one applicable transition.

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

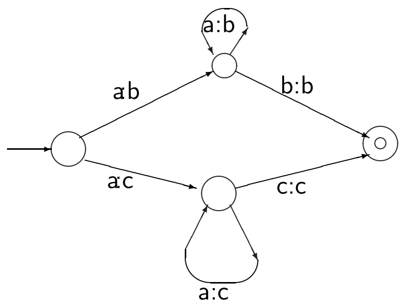
A **finite state transducer** is a 6-tuple $(Q, \Sigma_1, \Sigma_2, E, S, F)$ with

- 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:



Reading assignment

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