Ling 5801: Lecture Notes 1
Cognitive Models and Finite State Automata

Contents

1.1 Course goals

This course will cover three kinds of computational linguistics:

1. computation *in* language (modeling human comprehension)
2. computation *on* language (engineering solutions, information extraction)
3. computation *for* linguistics (intro programming)

We will start with computation *in* language

1.2 Background: some math notation (in case you don't know)

Set notation, involving **sets** S, S' and **entities** $x, x', x'', x_1, x_2, x_3, \ldots$:

pair	$\langle x_1, x_2 \rangle$
tuple	$\langle x_1, x_2, x_3, ... \rangle$
set	$S = \{x \mid ...\}$ e.g. $\{x_1, x_2, x_3\}$
empty/null set	$\emptyset$ or $\{\}$
element	$x \in S$ e.g. $x_2 \in \{x_1, x_2\}$, $x_3 \notin \{x_1, x_2\}$
subset (or equal)	$S \subset S'$ e.g. $\{x_1, x_2\} \subset \{x_1, x_2, x_3\}$, $\{x_1, x_2\} \subseteq \{x_1, x_2\}$
union	$S \cup S'$ e.g. $\{x_1, x_2\} \cup \{x_2, x_3\} = \{x_1, x_2, x_3\}$
intersection	$S \cap S'$ e.g. $\{x_1, x_2\} \cap \{x_2, x_3\} = \{x_2\}$
exclusion or complementation	$S - S'$ e.g. $\{x_1, x_2\} - \{x_2, x_3\} = \{x_1\}$
Cartesian product	$S \times S'$ e.g. $\{x_1, x_2\} \times \{x_3, x_4\} = \{\langle x_1, x_3 \rangle, \langle x_1, x_4 \rangle, \langle x_2, x_3 \rangle, \langle x_2, x_4 \rangle\}$
power set	$\mathcal{P}(S)$ or 2^S e.g. $\mathcal{P}(\{x_1, x_2\}) = \{\emptyset, \{x_1\}, \{x_2\}, \{x_1, x_2\}\}$
relation	$R \subseteq S \times S' = \{\langle x, x' \rangle \mid ...\}$ e.g. $R = \{\langle x_1, x_3 \rangle, \langle x_2, x_3 \rangle, \langle x_2, x_4 \rangle\}$

function	$F : S \to S' \subseteq S \times S'$ s.t. if $\langle x, x' \rangle, \langle x, x'' \rangle \in F$ then $x' = x''$
cardinality	$\lvert S \rvert$ = number of elements in S

First-order logic notation, involving **propositions** p, p' – e.g. that 1<2 (true) or 1=2 (false):

conjunction	$p \wedge p'$ or p, p' e.g. $1<2 \wedge 2<3$ or $1<2, 2<3$: both p and p'
disjunction	$p \vee p'$ e.g. $1<2 \vee 1>2$: either p or p'
negation	$\neg p$ or '/' e.g. $\neg 1=2$ or $1 \neq 2$: not p
existential quantifier	$\exists_{x \in S} \ldots x \ldots$: disjunction over all x of proposition $\ldots x \ldots$
universal quantifier	$\forall_{x \in S} \ldots x \ldots$: conjunction over all x of proposition $\ldots x \ldots$

Limit notation, involving **sets** S and **entities** x:

existential quantifier	$\bigvee_{x \in S} \ldots x \ldots$:	disjunction over all x of proposition $\ldots x \ldots$
universal quantifier	$\bigwedge_{x \in S} \ldots x \ldots$:	conjunction over all x of proposition $\ldots x \ldots$
limit union	$\bigcup_{x \in S} \ldots x \ldots$:	union over all x of set $\ldots x \ldots$
limit intersection	$\bigcap_{x \in S} \ldots x \ldots$:	intersection over all x of set $\ldots x \ldots$
limit Cartesian product	$\bigtimes_{x \in S} \ldots x \ldots$:	Cartesian product over all x of set $\ldots x \ldots$
limit sum	$\sum_{x \in S} \ldots x \ldots$:	sum over all x of number $\ldots x \ldots$
limit product	$\prod_{x \in S} \ldots x \ldots$:	product over all x of number $\ldots x \ldots$

1.3 Finite State Automata: a simple model of language as letter strings

Let's start with a simple model to recognize **the set of letter strings in a language**

For example, the set of overheard phone conversation sequences:

{**hello ok goodbye**, **hello ok ok goodbye**, **hello ok ok ok goodbye**, . . . }

(These sets may be infinite, but can still have various kinds of complexity limits!)

(We won't deal with the *meanings* of these strings yet, just the strings themselves.)

We can model this with a **Finite State Automaton/Machine (FSA)** $A = \langle Q_A, X_A, S_A, F_A, M_A \rangle$

- Q_A is a set of **states** (e.g. state of waiting for response / goodbye / verb phrase / . . .)
- X_A is a set of **observed symbols** (e.g. letters, words or phonemes)
- $S_A \subseteq Q_A$ is a subset of **start states** (e.g. state when you answer the telephone)
- $F_A \subseteq Q_A$ is a subset of **final states** (e.g. state when it's ok to hang up)
- $M_A \subseteq Q_A \times X_A \times Q_A$ is a **transition relation** or 'model' (e.g. 'hello' → 'ok')

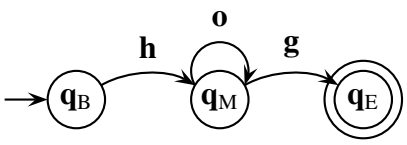
1.4 Graphical representation of FSAs

We can draw FSAs by representing:

- **states** as circles
- **transitions** as labeled arcs
- **observed symbols** as labels on arcs
- **start states** designated with short unlabeled arc
- **final states** designated with double circles

E.g. phone call: Q = conversation states (begin, middle, end); X = words (hello, ok, goodbye)

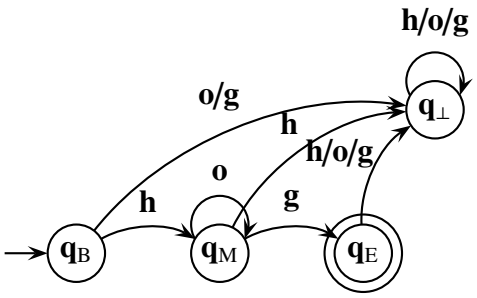
$Q{:}\{\mathbf{q}_B, \mathbf{q}_M, \mathbf{q}_E\},\ X{:}\{\mathbf{h}, \mathbf{o}, \mathbf{g}\},\ S{:}\{\mathbf{q}_B\},\ F{:}\{\mathbf{q}_E\},\ M{:}\{\langle \mathbf{q}_B, \mathbf{h}, \mathbf{q}_M\rangle, \langle \mathbf{q}_M, \mathbf{o}, \mathbf{q}_M\rangle, \langle \mathbf{q}_M, \mathbf{g}, \mathbf{q}_E\rangle\}\ =$



Like 'Candyland': states are board spaces, input $x_{1..T}$ is stack of cards (but, multiple pieces)

1.5 Sink states

We can define **sink states** $\mathbf{q}_\perp$ to give bad sequences a place to die.



These are often assumed, but not shown.

1.6 FSAs can define languages

Here, **strings** are **sequences of observations** $x_{1..T} = x_1, x_2, ..., x_{T-1}, x_T$.

Formally, the **set of strings** or **language** $L(A)$ recognized by an FSA A is:

$$L(A) = \{\ \underbrace{x_{1..T}}_{\text{set of observation sequences}}\ |\ \overbrace{\exists_{q_{0..T}}}^{\text{there is a state sequence}}\ \underbrace{q_0 \in S_A}_{\text{starts in a start state}}, \underbrace{q_T \in F_A}_{\text{ends in a final state}}, \overbrace{\forall_{t\in\{1,...,T\}}}^{\text{for each position}}\ \underbrace{q_t \in Q_A, x_t \in X_A, \langle q_{t-1}, x_t, q_t\rangle \in M_A}_{\text{state, observation, transition are in the FSA}}\}$$

The **set of languages** (so, the set of sets of strings) recognized by all FSAs is:

$\mathcal{L}(FSA) = \{L \mid \exists_{\text{FSA } A}\ L = L(A)\}$

We can also think of FSAs as **generating** strings $x_1 ... x_T$.

Some think of FSAs as **transducing** (generating) state sequences $q_1 ... q_T$ **from** observations $x_1 ... x_T$.

1.7 What's a state?

A state is a description of some aspect of the world; a proposition.

It summarizes information about past observations needed to process future observations.

As propositions, states in an FSA are mutually exclusive: only one at a time is true.

(Later we will define probability distributions over them.)

1.8 What's a final state?

It delimits a set of *acceptable* observation sequences (the strings in the language).

1.9 Example

FSA can recognize valid solutions to farmer river puzzle:

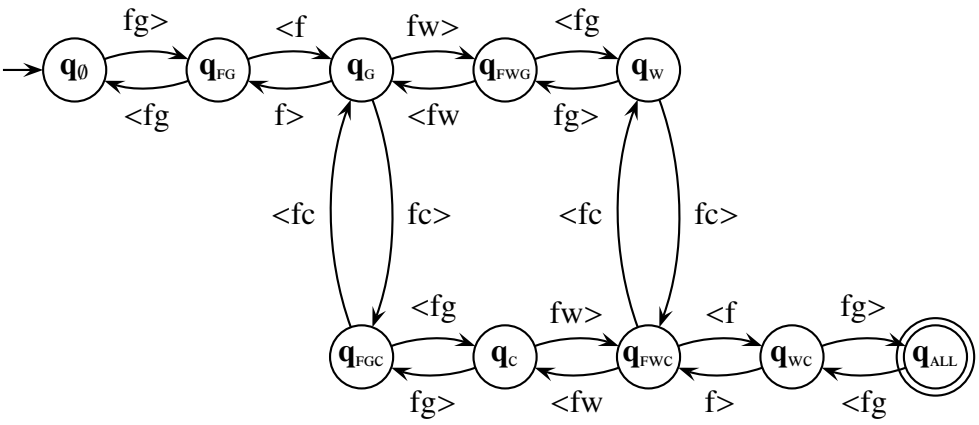
- farmer 'f', wolf 'w', goat 'g', cabbage 'c' want to cross river
- boat only holds two characters
- if farmer not present, wolf will eat goat / goat will eat cabbage

Observations are actions (farmer crossing river alone or with other character):

X_{FR} = { 'f>', 'fw>', 'fg>', 'fc>', '<f', '<fw', '<fg', '<fc' }

States are configurations of characters on right (and implicitly left) side of river + fail:

Q_{FR} = { $\mathbf{q}_{\emptyset}$, $\mathbf{q}_{\text{F}}$, $\mathbf{q}_{\text{W}}$, $\mathbf{q}_{\text{FW}}$, $\mathbf{q}_{\text{G}}$, $\mathbf{q}_{\text{WG}}$, $\mathbf{q}_{\text{FG}}$, $\mathbf{q}_{\text{FWG}}$, ..., $\mathbf{q}_{\text{ALL}}$ }



Phone conversation model makes predictions, so some transitions not used.

Farm river model evaluates solutions, so need all transitions: use 'sink' state.

Practice:

Design an FSA to interpret pet utterances.

Your pet starts off satisfied, then becomes hungry. It vocalizes every few seconds, burbling contentedly when satisfied, then whiffling crossly when it becomes hungry. It then stays hungry but may either whiffle or burble, since it knows you will feed it soon. Your FSA should detect whether your pet is hungry at the end of its utterance.

What elements would you use for Q, X, S, F, and M?

Draw your FSA.

Discussion:

How did you do? Anything unclear?

Are there other ways to extend model?

1.10 What does this have to do with human language?

An FSA is a simple system for recognizing sequences of observations over time.

It functions as a detector (e.g. of grammaticality)

Richer states can encode 'states of cognition,' with substates for syntax and meaning.

(In this case we don't really need final states.)

1.11 FSAs are also a nice way to get information from text

- grep
- sed
- perl
- unix wildcard
- word processors
- ...