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

1. Course goals:

(a) computation \textit{in} language (modeling human comprehension)
(b) computation \textit{on} language (engineering solutions, information extraction)
(c) computation \textit{for} linguistics (intro programming)

we will start with computation \textit{in} language

2. Background (in case you don’t know): set notation

\begin{itemize}
  \item pair \quad \langle q_1, q_2 \rangle
  \item tuple \quad \langle q_1, q_2, q_3, \ldots \rangle
  \item set \quad S = \{ q \mid \ldots \} \quad \text{e.g.} \quad \{ q_1, q_2, q_3 \}
  \item empty/null set \quad \emptyset \text{ or } \{ \}
  \item element \quad q \in S \quad \text{e.g.} \quad q_2 \in \{ q_1, q_2 \}, \quad q_3 \notin \{ q_1, q_2 \}
  \item subset (or equal) \quad S \subseteq S' \quad \text{e.g.} \quad \{ q_1, q_2 \} \subseteq \{ q_1, q_2, q_3 \}, \quad \{ q_1, q_2 \} \subseteq \{ q_1, q_2 \}
  \item union \quad S \cup S' \quad \text{e.g.} \quad \{ q_1, q_2 \} \cup \{ q_2, q_3 \} = \{ q_1, q_2, q_3 \}
  \item intersection \quad S \cap S' \quad \text{e.g.} \quad \{ q_1, q_2 \} \cap \{ q_2, q_3 \} = \{ q_2 \}
  \item exclusion \quad S - S' \quad \text{e.g.} \quad \{ q_1, q_2 \} - \{ q_2, q_3 \} = \{ q_1 \}
  \item Cartesian product \quad S \times S' \quad \text{e.g.} \quad \{ q_1, q_2 \} \times \{ q_3, q_4 \} = \{ (q_1, q_3), (q_1, q_4), (q_2, q_3), (q_2, q_4) \}
  \item power set \quad \mathcal{P}(S) \text{ or } 2^S \quad \text{e.g.} \quad \mathcal{P}(\{ q_1, q_2 \}) = \{ \emptyset, \{ q_1 \}, \{ q_2 \}, \{ q_1, q_2 \} \}
  \item relation \quad R \subseteq S \times S' = \{ \langle q, q' \rangle \mid \ldots \} \quad \text{e.g.} \quad R = \{ \langle q_1, q_2 \rangle, \langle q_1, q_3 \rangle, \langle q_3, q_4 \rangle \}
  \item function \quad F \subseteq S \rightarrow S' \subseteq S \times S' \text{ s.t. if } \langle q, q' \rangle, \langle q, q'' \rangle \in F \text{ then } q' = q''
  \item cardinality \quad |S| = \text{number of elements in } S
\end{itemize}

Background (in case you don’t know): first-order logic notation

\begin{itemize}
  \item conjunction \quad o \land o' \quad \text{or} \quad o \land o' \quad \text{e.g.} \quad 1 < 2 \land 2 < 3 \quad \text{or} \quad 1 < 2, 2 < 3
  \item disjunction \quad o \lor o' \quad \text{e.g.} \quad 1 < 2 \lor 1 > 2
  \item negation \quad \neg o \quad \text{or} \quad ' \neg' \quad \text{e.g.} \quad \neg 1 = 2 \quad \text{or} \quad 1 \neq 2
  \item existential quantifier \quad \exists x \in X \ldots : \text{disjunction over all } x \text{ of } \ldots x \ldots
  \item universal quantifier \quad \forall x \in X \ldots : \text{conjunction over all } x \text{ of } \ldots x \ldots
\end{itemize}

Background (in case you don’t know): limit notation

\begin{itemize}
  \item existential quantifier \quad \bigvee_{x \in X} \ldots : \text{disjunction over all } x \text{ of } \ldots x \ldots
  \item universal quantifier \quad \bigwedge_{x \in X} \ldots : \text{conjunction over all } x \text{ of } \ldots x \ldots
  \item limit union \quad \bigcup_{x \in X} \ldots : \text{union over all } x \text{ of } \ldots x \ldots
  \item limit intersection \quad \bigcap_{x \in X} \ldots : \text{intersection over all } x \text{ of } \ldots x \ldots
  \item limit sum \quad \sum_{x \in X} \ldots : \text{sum over all } x \text{ of } \ldots x \ldots
  \item limit product \quad \prod_{x \in X} \ldots : \text{product over all } x \text{ of } \ldots x \ldots
  \item limit Cartesian product \quad \bigtimes_{x \in X} \ldots : \text{Cartesian product over all } x \text{ of } \ldots x \ldots
\end{itemize}
3. In language: 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.)

Finite State Automaton/Machine (FSA) \( A = (Q_A, X_A, S_A, F_A, M_A) \)

- \( Q_A \) 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’ \( \rightarrow \) ‘ok’)

4. Graphical representation of FSA:

- states are circles
- transitions are labeled arcs
- observed symbols are 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)

\[ Q : \{ q_B, q_M, q_E \}, \ X : \{ h, o, g \}, \ S : \{ q_B \}, \ F : \{ q_E \}, \ M : \{ \langle q_B, h, q_M \rangle, \langle q_M, o, q_M \rangle, \langle q_M, g, q_E \rangle \} = \]

Like ‘Candyland’: states are board spaces, input \( x_{1:T} \) is stack of cards (but, multiple pieces)

5. Sink state \( q_\perp \): usually not shown, gives bad sequences a place to die
6. FSAs can recognize strings (sequences of observations \( x_{1:T} = x_1, x_2, ..., x_{T-1}, x_T \))

set of strings (or ‘language’) \( L(A) \) recognized by an FSA \( A \):

\[
L(A) = \{ x_{1:T} \mid \exists q_0, q_T \in S_A, q_1, q_T \in F_A, \forall 1 \leq t \leq T \rightarrow q_t \in Q_A, x_t \in X_A, (q_{t-1}, x_t, q_t) \in M_A \}
\]

set of languages recognized by any FSA:

\[
\mathcal{L}(FSA) = \{ L \mid \exists FSA A : L = L(A) \}
\]

can also think of FSA as generating strings \( x_1...x_T \)

can also think of FSA as transducing (generating) state strings \( q_1...q_T \) from obs \( x_1...x_T \)

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)

8. What’s a final state?

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

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):

\[ X_{FR} = \{ \text{‘f’}, \text{‘fw’}, \text{‘fg’}, \text{‘fc’}, \text{‘<f’}, \text{‘<fw’}, \text{‘<fg’}, \text{‘<fc’} \} \]

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

\[ Q_{FR} = \{ q_0, q_f, q_w, q_{fw}, q_g, q_{wg}, q_{fg}, q_{fwg}, ..., q_{all} \} \]

---

3
Phone conversation model makes predictions, so some transitions not used.
Farm river model evaluates solutions, so need all transitions: use ‘sink’ state.

10. Question: design an FSA to interpret pet utterances. Your pet starts off satisfied, then becomes hungry. It vocalizes every few seconds, purring/panting/cooing/twittering contentedly when satisfied, barking/meowing/oinking/shrieking when it becomes hungry, then stays hungry whatever its vocalization. The FSA should detect whether the pet is hungry at the end of its utterance.

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

Draw your FSA.

11. Discussion: how did you do? Anything unclear?
Are there other ways to extend model?

12. What does this have to do with human language?
FSA is a simple system for recognizing sequences of observations over time functions as a detector (e.g. of grammaticality)

richer states can encode ‘states of cognition,’ with substates for syntax and meaning (then, don’t really need final states)

13. Also a nice way to get information from text
   
   • grep
   • sed
   • perl
   • unix wildcard
   • word processors
   • ...