Today's lecture: Everything You Ever Wanted to Know About Finite Automata

Reading:

Today 30: Survey of other topics related to finite automata
Read: *Kozen* lecture 13

October 3: Context Free Languages and Grammars
Read: *Kozen* lecture 19 or *Sipser* 2.1.

October 5: Balanced Parentheses
Read: *Kozen* lecture 20 (or *Sipser* 2.1).

October 7: Chomsky and Greibach normal forms
Read: *Kozen* lecture 21 (or *Sipser* 2.1).

October 10: Thanksgiving, no lecture

October 12: Non-Context-Free Languages
Read: *Kozen* lecture 22 or *Sipser* 2.3.

October 14: Non-deterministic, Pushdown Automata
Read: *Kozen* lecture 23 or *Sipser* 2.2.

October 17: From CFLs to PDAs
Read: *Kozen* lecture 24 (or *Sipser* 2.2).

October 19: From PDAs to CFLs
Read: *Kozen* lecture 25 (or *Sipser* 2.2).

October 21: Deterministic PDAs
Read: *Kozen* lectures E and F.

October 24: Parsing
Read: *Kozen* lecture 26 (not in Sipser)

October 26: Midterm: In class.

October 28: A Parsing Algorithm
Read: *Kozen* lecture 27 (not in Sipser)

October 31: LALR Parsing
Read: TBD

I. Equivalent Automata

A. How can we tell if two regular languages are the same?

1. Let B_1 and B_2 be two regular languages.

a. Let $M_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$ be a DFA that recognizes B_1.

b. Let $M_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, F_2)$ be a DFA that recognizes B_2.

c. Note that B_1 and B_2 could be described initially as DFAs, NFAs, REs, or Kozen's pattern – we know how to convert any of these to DFAs.

2. Let $M_{12} = (Q_{12}, \Sigma, delta_{12}, q_{0,12}, F_{12})$ be the product machine for $M_1 \times M_2$.

a. In particular:

$$\begin{array}{rcl} Q_{12} & = & Q_1 \times Q_2 \\ \delta_{12}((q_1, q_2), \mathtt{c}) & = & (\delta_1(q_1, \mathtt{c}), \delta_2(q_2, \mathtt{c})) \\ q_{0,12} & = & (q_{0,1}, q_{0,2}) \\ F_{12} & = & (F_1 \times \sim F_2) \cup (\sim F_1 \times F_2) \end{array}$$

b. The language of M_{12}

$$\begin{array}{ll} & w \in L(M_{12}) \\ \Leftrightarrow & \hat{\delta}_{12}(q_{0,12}, w) \in F_{12}, \\ \Leftrightarrow & (\hat{\delta_1}(q_{0,1}, w), \hat{\delta_1}(q_{0,1}, w)) \in (F_1 \times \sim F_2) \cup (\sim F_1 \times F_2), \\ \Leftrightarrow & \left((\hat{\delta_1}(q_{0,1}, w) \in F_1) \wedge (\hat{\delta_1}(q_{0,1}, w) \in \sim F_2)\right) \vee \left((\hat{\delta_1}(q_{0,1}, w) \in \sim F_1) \wedge (\hat{\delta_1}(q_{0,1}, w) \in F_2)\right), \\ \Leftrightarrow & (\hat{\delta_1}(q_{0,1}, w) \in \sim F_1) \neq (\hat{\delta_1}(q_{0,1}, w) \in F_2), \text{Boolean algebra} \\ \Leftrightarrow & w \in B_1 \ominus B_2, \end{array}$$

def

Ko

set

choice of M_1 and M_2

where $B_1 \ominus B_2$ denotes symmetric set difference – $B_1 \ominus B_2 \;=\; (B_1 - B_2) \cup (B_2 - B_1)$. In English, M_{12} accepts a string, w, iff it is in one of B_1 or B_2 but not both. Such a string is a *witness* that B_1 and B_2 are different languages.

3. B_1 and B_2 are the same if $B_1 \ominus B_2 = \emptyset$.

a. To test this, construct M_{12} as above.

b. Verify that M_{12} has no accepting states that are reachable from the initial state.

c. This is a simple graph search problem.

i. The states in Q_{12} are the vertices of the graph.

ii. There is a directed edge from q_1 to q_2 iff there exists some symbol c such that $q_2 = \delta_{12}(q_1, \mathtt{c})$

4. Summary: to test if B_1 and B_2 are the same language.

a. Construct DFAs for B_1 and B_2. Call them M_1 and M_2.

b. Construct a product automaton M_{12} that accepts iff B_1 accepts and B_2 doesn't or B_2 accepts and B_1 doesn't.

c. Make sure that no accepting states of M_{12} are reachable from the initial state. This can be done with your favorite traversal algorithm for directed graphs.

B. Applications of equivalence

1. We can specify network protocols as finite automata.

2. The code that implements a protocol can be modeled as a finite automaton.

3. Because the automata for the specification and implementation where derived separately, the correspondence may not be immediately obvious.

4. We can use the construction described above to determine whether or not the software correctly implements the protocol.

a. If it doesn't we find a string w that is in $B_{spec} \ominus B_{impl}$. This is a string that gives a counter-example – it demonstrates the bug.

b. Sometimes, the specification is not the full, detailed automata, but instead a set of properties that the implementation should have. We can often model each property with a regular language. Let's call these P_1, P_2, $\ldots P_k$. Now, to make sure that our implementation has all of the required properties, we check $B \in P_i$ for each property P_i that we want to verify, where B is the automaton modeling our implementation.

5. This approach to verification is called "model checking" and has become an important part of network protocol design and implementation and hardware verification. There is a growing interest in software model checking, where properties of software systems are modeled by automata and these kinds of checking methods are applied.

C. What is the smallest DFA that accepts a particular language?

- **a.** First, we eliminate unreachable states.
- **b.** Second, we collapse "equivalent" states into a single state
  - **i.** Two states of a DFA are equivalent, iff the sets of strings that lead to an accepting state are the same for the two states.
  - **ii.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. To check if state q_i and q_j are equivalent, construct automata

$$\begin{array}{rcl} M_i & = & (Q, \Sigma, \delta, q_i, F) \\ M_j & = & (Q, \Sigma, \delta, q_j, F) \end{array}$$

    Note that M_i accepts string w if the path starting at state q_i reading w leads to an accepting state. Likewise for M_j and q_j.
    If M_i and M_j are equivalent, we say that q_i and q_j are equivalent.
  - **iii.** We can collapse equivalent states into a single state. No matter how the original automaton got to any of these states, the suffixes of w that will lead to an accepting state are the same for all of them. This doesn't change the language accepted by the machine.
- **c.** Kozen formalizes all of this in chapter 13.

D. The uniqueness of being small.
Let M_1 and M_2 be two machines that accept the same language and that have both been minimized by the method described above.

- **a.** If $\hat{\delta}_1(q_{1,0}, w_1) = \hat{\delta}_1(q_{1,0}, w_2)$ then $\hat{\delta}_2(q_{1,0}, w_1) = \hat{\delta}_2(q_{1,0}, w_1)$.
- **b.** This means that M_1 and M_2 are equivalent to within renaming the states.

II. Other versions of finite automata

A. Automata on trees

B. 2-way DFAs

C. Automata on infinite strings

D. Automata for reactive systems

- **1.** predicates as input symbols
- **2.** timed automata
- **3.** hybrid automata

E. Quantum Automata

III. A pumping lemma example: the prime number language from the Sept. 26 notes.