Lecture 25. Finite Automata

Reading:
Chapter 17, "Finite Automata, Regular Languages, and Type 3 Grammars" of PtMW, just Section 17.1: Finite Automata. pp. 453-462.

Homework:
Homework 25 on syllabus.

Erratum to PtMW p. 464: the single state q_0 in Figure 17-5 a) is a final state as well as an initial state, and should therefore be drawn with a double circle, not a single circle.

17.1 Finite automata.

Terminology: *finite automaton = finite state automaton*. The class is sometimes called FSA, or just FA. The class of languages accepted by a fsa is called a *finite state language.* These are also the *regular languages*, but we use a separate definition, later, to define 'regular language', and then we prove the equivalence of the two classes. We will also define a particular class of formal grammars, the Type 3 grammars, and we can prove that the class of languages definable by Type 3 grammars is exactly the same class, the regular languages. We will thus have three independent characterizations of the same class of languages.

An automaton may be *deterministic* or *non-deterministic.* We will first define deterministic fsa, then non-deterministic, then show that for fsa (this is not true for some other classes of automata), the two subclasses of fsa are equivalent with respect to the class of languages accepted.

State diagrams. Example illustrating how dfa work. [blackboard; fig 17-2, p.456]

Formal definition:
Definition 17.1. A deterministic finite automaton (dfa) M is a 5-tuple $<K, \Sigma, \delta, q_0, F>$, where
K is a finite set, the set of states
Σ is a finite set, the alphabet
$q_0 \in K$, the initial state
$F \subseteq K$, the final states
δ is a function from $K \times \Sigma$ into K, the transition function (or next-state function).

What makes this a *deterministic* fsa is that δ must be a function: for each state and symbol, there is exactly one transition to a next state.

Skipping in class: the formal definition of a "situation", a binary relation "produces-in-one-move", and the defined binary relation "produces-in-zero-or-more-moves".

Then we can say that a dfa M *accepts* a string x in Σ^* iff M produces x in zero or more moves. The *language* L(M) *accepted by* a dfa M is the set of all strings accepted by M.

Non-deterministic fa's (nfa).
Two in-principle weakenings of the requirements, and two more that are 'optional' but commonly included.
(i) for a given state-symbol pair, possibly more than one next state. [this is THE crucial one]
(ii) for a given state-symbol pair, possibly no next state. [this could always be modelled by adding a "dead-end state"]
(iii) allowing a transition of the form (q_i, w, q_j) where $w \in \Sigma^*$, i.e. being able to read a string of symbols in one move, not only a single symbol. And as a noteworthy subcase of that,
(iv) allowing a transition of the form (q_i, e, q_j) : changing state without reading a symbol.

Example. fig 17-3, p. 459

An input tape is *accepted* by a non-deterministic fa if there is *some* path through the state diagram which begins in the initial state, reads the entire string, and ends in a final state.

Formal definition of non-deterministic fa. Just like formal definition of dfa, except that in place of the transition <u>function</u> δ there is a transition <u>relation</u> Δ, a finite subset of $K \times \Sigma^* \times K$.
The definitions of acceptance of a string, and of accepted language, are exactly analogous to the earlier ones.

Equivalence of deterministic and non-deterministic fsa. This is a major result – it is not self-evident. The algorithm for constructing an equivalent deterministic fsa, given a non-deterministic one, is a bit complex and we won't do it; in the worst case it may give a dfa with 2^n states corresponding to a nfa with n states. (And that presupposes that we take the narrower definition of nfa, with weakenings (i) and (ii) but not (iii) or (iv).)

Why it is useful to have both notions: The deterministic fa are conceptually more straightforward; but in a given case it is often easier to construct a non-deterministic fa. Also, for some other classes of automata that we will consider, the two subclasses are not equivalent, so the notions remain important.