CS 42—Minimality and Nondeterministic Finite Automata (NFAs)

Tuesday, September 11, 2018

Summary

Our big questions right now is: **What do we mean by "computer"?**

Last week, we introduced a model of computation called Deterministic Finite Automata (DFAs), also known as Finite State Machines (FSMs).

Today, we'll evaluate DFAs by exploring what makes for a "good" DFA, and by asking "What are DFAs good for?"

Reminder: How DFAs work

A DFA has a finite alphabet (Σ) of valid symbols.

A DFA has a finite set of states—one initial state and some accepting states.

A DFA has a transition function. The transition function maps every possible *configuration* to a state. A configuration consists of two parts: a state and an input symbol. The transition function means that—given a state and an input symbol—there is always one and only one transition that the machine can take. This property is what makes it "deterministic".

Each transition consumes one input symbol.

The machine can be in exactly one state at a time.

Given an input string, a DFA operates as follows:

> The machine starts in the initial state.
>
> The machine takes the only applicable transition (consuming an input symbol) until it has read and responded to the entire input string. When all the input is consumed, the machine halts.
>
> The machine accepts only if it is in an accepting state.

Minimality

Given a language L, can we *prove* that a DFA for L requires at least n states, for some n?

Definition: Two strings w_1, w_2 are **distinguishable** if there is some other string z such that $w_1 z \in L$ and $w_2 z \notin L$.

Definition: A set of strings S is **pairwise-distinguishable** for a language L if every pair of strings $w_i \neq w_j$ is distinguishable.

Theorem: If a set of strings $S = \{w_1, w_2, \ldots, w_n\}$ is pairwise distinguishable for a language L, then any DFA that accepts L must have at least n states.

Nondeterministic Finite Automata (NFAs)

An NFA has a finite alphabet (Σ) of valid symbols.

An NFA has a finite set of states—one initial state and some accepting states.

An NFA has a transition relation. The transition relation maps a configuration to zero or more states. If a configuration does not map to any states, we assume there is a transition from that configuration to an implicitly defined state that rejects ever after. (In other words, if the machine is in a particular state and there aren't any transitions defined for the current input symbol, then the machine rejects the input.)

Each transition consumes at most one input symbol. A special transition—called a **λ-transition**—changes state, but consumes no input. All other transitions consume one input symbol.

The machine can be in multiple states at the same time.

Given an input string, an NFA operates as follows:

> The machine starts in the initial state.
>
> The machine takes *all* applicable transitions until it has read and responded to the entire input string. When all the input is consumed, the machine halts.
>
> The machine accepts only if it is in at least one accepting state.

Next time: Are DFAs what we mean by "computer"? Are NFAs?