CS 42—Deterministic Finite Automata (DFAs)

Thursday, September 6, 2018

Summary

Today is the first of a series of three classes where we explore the question:
What kinds of problems can computers solve?

This question raises another one, which we'll continue to explore for the next three weeks:
What do we mean by computer?

A brief refresher on binary

A *bit* is a binary digit: 0 or 1.

A *bitstring* is a sequence of zero or more bits.

By convention, we use the Greek letter λ (pronounced "lambda") to represent an empty string (i.e., a string of length 0). *[Note: People also use ε (pronounced "epsilon") for the empty string.]*

A bitstring's *most-significant digit* is the left-most digit; it's *least-significant digit* is the right-most.

We can assign a *numeric value* to non-empty bitstrings by relating each digit to a power of two. The least significant digit corresponds to 2^0. The value for that digit is either $2^0 * 0 = 0$ or $2^0 * 1 = 1$.

0	0	**1**	0	**1**	0	**1**	0	
2^7	2^6	$\mathbf{2^5}$	2^4	$\mathbf{2^3}$	2^2	$\mathbf{2^1}$	2^0	
128	64	**32**	16	**8**	4	**2**	1	
		32	+	**8**	+	**2**	=	42

What kinds of problems can computers solve?

To answer this question, we need better definitions of "problem" and "computer". We're going to try to define these two terms in most precise way we can, finding a *formal model* for both of them.

A formal model is a precise (usually mathematical) *abstraction* of the thing we want to talk about. It should be as general as possible (e.g., it should fit all our notions of what a "problem" is or what a "computer" is), while also being as simple as possible to state (i.e., there aren't very many pieces to the formal model).

What do we mean by “problem"? What do we mean by “computer”?

Our formal model for “problem” will be *decision problems*—questions with yes or no answers.

Today, our formal model for “computer” will be *state machines*.

State machine

Reads the **input** string one symbol at a time.

Has a set of possible “configurations” (**states**).

Has rules for how to **transition** from one state to another, based on current state and current input symbol.

Accepts ("yes") or **rejects** ("no"), based on the input so far.

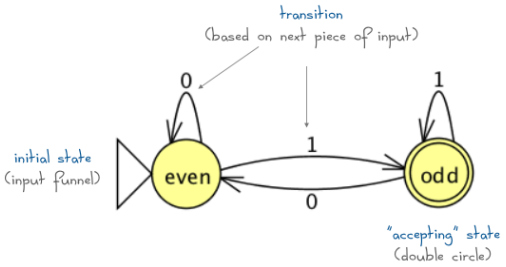
Finite state machines (FSMs) / Deterministic Finite Automata (DFAs)

A **finite state machine (FSM)** is a state machine with:

- a predetermined, finite-sized set of **states**
- a predetermined, finite-sized **alphabet Σ** of valid input characters
- a set of rules that describe **transitions** from each state for each character, so that each state knows what to do for any possible input character
- a designated **initial state**
- a set of accepting **states**

Finite state machines are also called Deterministic Finite Automata (DFAs). Engineers tend to use the former term; theoretical computer scientists tend to use the latter. We’ll use both terms, but we’ll use DFA more frequently.

A *finite* state machine is a restricted version of the state machine described in the previous section. An FSM has a finite number of states and stops running after it has read all its input. After it stops running, the machine accepts if it is in one of its predetermined set of accepting states.



DFAs define a language

A DFA’s *language* is the set of strings accepted by the DFA.

Next time: Proving properties of DFAs, more computational models