Structural induction and DFA union lecture summary

M. George

October 20, 2014

1 Addenda to last lecture

Pumping lemma In the statement of the pumping lemma, we can add the fact that $|uv| \leq m$ (where m is the number of states in the machine). This can be useful in proofs that use the pumping lemma; for example we could simplify the proof that $\{0^n 1^n \mid n \in \mathbb{N}\}$ is not regular by using the fact that $|uv| \leq m$ in the string $0^m 1^m$; this forces v to only contain zeros.

The proof of the extended lemma is exactly the same as the proof of the lemma we did last time.

Halting problem Last time we gave a non-constructive proof that there are undecidable languages. One famous example of such a language is the "Halting Problem", defined as follows (in the context of Java programs). The language L consists of all strings that, when interpreted as the source code of Java programs, halt.

It turns out this language cannot be decided by any Java program that always terminates.

2 Inductive definitions and structural induction

It is often useful to define a set using a set of rules for adding things to the set. For example, we can define the set Σ^* using the following rules:

1. $\epsilon \in \Sigma^*$ (ϵ is the empty string)
2. For all $x \in \Sigma^*$, and for all $a \in \Sigma$, xa is also in Σ^*.

We can then take Σ^* to be the smallest set that satisfies these rules. A set constructed this way is called an *inductively defined set*.

If a set S is inductively defined, we can define functions using the inductive rules. In the string example, every string is either ϵ or xa for some x and a; If I give you the definition of f for ϵ and also for xa (possibly in terms of $f(x)$)

then you can use that definition to evaluate f on any input. See the definition of $\hat{\delta}$ below for a concrete example.

For another example, consider the length function ℓ on strings. I will define $\ell(\epsilon) = 0$ and $\ell(xa) = 1 + \ell(x)$. This function is well defined. For example I can evaluate it on the string 101 as follows:

$$\ell(101) = 1 + \ell(10) = 1 + (1 + \ell(0)) = 1 + (1 + (1 + \ell(\epsilon))) = 3$$

Finally, if we have an inductively defined set, we can use *structural induction* to prove facts about all elements in the set. Just as with induction over the natural numbers, we need to include a case for each possible element of the set; which means we need to include a case for each of the rules that can be used to construct an element. We could for example prove that all strings have non-negative length:

Claim: For all $x \in \Sigma^*$, $\ell(x) \geq 0$.

Proof: There are two kinds of strings: ϵ and xa. By definition, $\ell(\epsilon) = 0 \geq 0$. We know $\ell(xa) = 1 + \ell(x)$. Our inductive hypothesis says that $\ell(x) \geq 0$. Adding one to both sides, we see that $\ell(xa) = 1 + \ell(x) \geq 1 \geq 0$.

3 Formal definition of a language

We then introduced some formal notation for proving things about DFAs. We defined the *extended transition function* $\hat{\delta} : Q \times \Sigma^* \to Q$. This function is extends the normal transition function δ for a DFA to accept entire strings instead of single characters. It is defined as follows:

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, xa) &= \delta(\hat{\delta}(q, x), a) \end{aligned}$$

Do not memorize this definition. You should get to a point where you can work it out from the high-level description of what it does.

Note that in the second line, one of the δ's has a hat and the other does not. These are both important; the first delta must not have a hat, because otherwise this is a circular definition (note that it wouldn't depend at all on δ), and the second delta must have a hat because δ cannot operate on entire strings.

Using this definition, we formalized the notion of a machine accepting a string x: x is accepted if $\hat{\delta}(q_0, x) \in F$. This gives us a definition for the language of a machine $M = (Q, \Sigma, \delta, F, q_0)$:

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$$

This is just a formalization of the notion of acceptance we have been using.

4 Union construction

We also constructed a machine to recognize the union of two DFA-recognizable languages. For more detail, see `constructions.pdf`.