CS 373: Theory of Computation (Summer 2011)
Lucas Cook

Lecture 14: DFA Minimization

2011 July 07 at 9:00 AM

---

1 Resources

For this lecture and the previous one, you probably cannot find detailed notes in the Sisper text. Try looking in the Hopcroft et al text (which provides an equivalent perspective using equivalence relations), or in the Fall 2010 lectures (from which we borrow the suffix language notation)[1].

2 Lecture 13 recap: Suffix languages

Last lecture, we observed that the states of a DFA partition the strings of Σ^*. Each string in Σ^* moves a DFA from q_0 to exactly one state $q \in Q$, and the thus we can group the strings by state. Furthermore, we claimed that if two strings w and w' lead to the same state, they 'behave the same' from then onwards in the DFA; for any additional input $z \in \Sigma^*$, wz and $w'z$ both accept or both reject.

We defined a *suffix language* of a string $w \in \Sigma^*$ with respect to a language L as

$$\text{Suffix}_L(w) = \{z \in \Sigma^* \mid wz \in L\}$$

and claim that two strings 'behave the same' when $\text{Suffix}_L(w) = \text{Suffix}_L(w')$. We also defined the *suffix class* of a language L as

$$\mathcal{C}_{\text{Suffix}}(L) = \{\text{Suffix}_L(w) \mid w \in \Sigma^*\}$$

the collection of suffix languages with respect to L.

For the regular language $L_{ex} = \mathcal{L}(1\Sigma^* \cup 0^+)$, we found that $\mathcal{C}_{\text{Suffix}}(L_{ex}) = \{L_{ex}, \Sigma^*, \mathcal{L}(0^*), \varnothing\}$, where each of these suffix languages corresponded to a state of the DFA for L_{ex}. This led us to the observation:

Remark 2.1 If L is a regular language, then $\mathcal{C}_{\text{Suffix}}(L)$ is finite.

We argued for this informally, noting that the states partition the strings, and each state will have one suffix language for all the strings that lead to it. Since there are a finite number of states, there are a finite number of suffix langauges.

For the nonregular language L_{0n1n}, we proved that $\mathcal{C}_{\text{Suffix}}(L_{0n1n})$ is infinite, by finding a different suffix language for each $w \in \mathcal{L}(0^*)$. This led us to the hypothesis:

[1] `http://www.cs.uiuc.edu/class/fa10/cs373/Lectures`

Claim 2.2 *L is a regular language if and only if $\mathcal{C}_{\text{Suffix}}(L)$ is finite.*

To prove the reverse ($\Leftarrow$) direction, we constructed $M_{\text{Suffix}} = (Q, \Sigma, \delta, q_0, F)$ using finite $\mathcal{C}_{\text{Suffix}}(L)$ as follows:

- $Q = \mathcal{C}_{\text{Suffix}}(L)$. Each state corresponds to a suffix language.
- $q_0 = \text{Suffix}_L(\varepsilon)$. The suffix language for no input (ε) is the start state.
- $\forall S \in Q, a \in \Sigma, \delta(S, a) = \text{Suffix}_L(wa)$, where w is some string with $\text{Suffix}_L(w) = S$. Transitions occur from one suffix language to another as determined by the strings that have those suffix languages.
- $F = \{S \in Q \mid \varepsilon \in S\}$. If ε is an accepted suffix, than this must be a final state.

We ended the lecture while proving that δ is a well defined transition function. So far, we argued that for any $S \in Q$, there is a w such that $\text{Suffix}_L(w) = S$ by the definition of $\mathcal{C}_{\text{Suffix}}(L)$.

3 M_{Suffix} correctness

Let's complete the argument that M_{Suffix} is a correct DFA and accepts L.

3.1 δ is a function

We need to show that δ behaves like a function from $Q \times \Sigma$ to Q. There is a problem in the choice of w: if we choose w and w' such that $w \neq w'$ but $\text{Suffix}_L(w) = \text{Suffix}_L(w')$, than we have to be sure that they lead the DFA to the same state, i.e. $\text{Suffix}_L(wa) = \text{Suffix}_L(w'a)$ for all $a \in \Sigma$. If this didn't happen, then δ could transition to different states on the same input (which isn't allowed in a DFA).

Lemma 3.1 *For strings $w, w' \in \Sigma^*$ and language L, if $\mathit{Suffix}_L(w) = \mathit{Suffix}_L(w')$, then $\mathit{Suffix}_L(wx) = \mathit{Suffix}_L(w'x)$ for all $x \in \Sigma^*$.*

Note that this claim is a generalization of what we need. Our case is where L has finite $\mathcal{C}_{\text{Suffix}}(L)$ and x is a single symbol a.

Proof: By contradiction, say that $\text{Suffix}_L(w) = \text{Suffix}_L(w')$ but $\text{Suffix}_L(wx) \neq \text{Suffix}_L(w'x)$ for some x. Then there must be $z \in \Sigma^*$ such that z is in exactly one of $\text{Suffix}_L(wx)$ and $\text{Suffix}_L(w'x)$. Without loss of generality, say $z \in \text{Suffix}_L(wx)$ and $z \notin \text{Suffix}_L(w'x)$. Thus $wxz \in L$ and $w'xz \notin L$, meaning that $xz \in \text{Suffix}_L(w)$ but $xz \notin \text{Suffix}_L(w')$. But this would mean $\text{Suffix}_L(w) \neq \text{Suffix}_L(w')$. ■

3.2 Correctness

Now let's argue that M_{Suffix} actually accepts L. For input $w = w_1 w_2 w_3 \ldots w_n \in \Sigma^*$, M_{Suffix} runs through the state sequence

$$\text{Suffix}_L(\varepsilon), \text{Suffix}_L(w_1), \text{Suffix}_L(w_1 w_2), \text{Suffix}_L(w_1 w_2 w_3), \ldots \text{Suffix}_L(w_1 w_2 w_3 \ldots w_n)$$

by the definition of the transition function.

If $w \in L$, then the sequence should accept at the end. Since $w \circ \varepsilon \in L$, then $\varepsilon \in \text{Suffix}_L(w)$ and $\text{Suffix}_L(w) \in F$. Thus $\text{Suffix}_L(w) = \text{Suffix}_L(w_1 w_2 w_3 ... w_n)$ is an accept state.

Similarly, if $w \notin L$, then the sequence should reject. Since $w \circ \varepsilon \notin L$, then $\varepsilon \notin \text{Suffix}_L(w)$ and $\text{Suffix}_L(w) \notin F$. Thus $\text{Suffix}_L(w) = \text{Suffix}_L(w_1 w_2 w_3 ... w_n)$ is a reject state.

4 Myhill-Nerode Theorem

Claim 2.2 is telling us that a (somewhat obscure) structural property $\mathcal{C}_{\text{Suffix}}(L)$ of a language determines its regularity or nonregularity. This is generally known as the Myhill-Nerode theorem, though usually through different formalism.

Theorem 4.1 ***(Myhill-Nerode Theorem)***
For any language L,
(<u>our version</u>) L is regular iff $\mathcal{C}_{\text{Suffix}}(L)$ is finite. Furthermore, there is a unique (upto isomorphism) minimal DFA for L determined by $\mathcal{C}_{\text{Suffix}}(L)$.
(<u>typical version</u>) L is regular iff there is an equivalence relation $\sim_L$ on strings such that

$$x \sim_L y \text{ when } \forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L$$

and $\sim_L$ has finite index. Furthermore, there is a unique (upto isomorphism) minimal DFA for L determined by $\sim_L$.

We've seen the first part, which is the same as Claim 2.2. The second part is interesting though: there is a unique optimal 'program' to solve any finite memory decision problem. It turns out that this is a consequence of Claim 2.2, and M_{Suffix} is the unique, minimal DFA! Even better, there is a systematic technique (i.e. algorithm) for generating the minimal DFA given any DFA for the language.

4.1 M_{Suffix}: minimal and unique

For a DFA to be *minimal*, there cannot be a DFA with fewer states that accepts the same language. (Intuitively, this is the machine that uses the least memory to solve the problem.) The claim is that M_{Suffix} is the minimal DFA for $L = \mathcal{L}(M_{\text{Suffix}})$. In previous lectures, we argued informally that DFA states must correspond to a suffix language. Thus any DFA must have at least $|\mathcal{C}_{\text{Suffix}}(L)|$ states to account for each suffix language. [2]

For a DFA to be *unique*, it must be the only one with its number of states. Note though that we can easily make an infinite number of DFAs with the same state size just by renaming states. (e.g. state sets $Q = \{a, b, c, d\}$ and $Q' = \{1, 2, 3, 4\}$ technically are different, even though their edges might look the same.) In this context, we only want structural differences to matter. If two DFAs are the same except for renaming the state labels, we say they are

[2]The formal proof of this in the Fall 2010 lectures proceeds as follows: for a DFA $M = (Q, \Sigma, \delta, q_0, F)$, a certain mapping $f : Q \to \mathcal{C}_{\text{Suffix}}(\mathcal{L}(M))$ between any DFA's states and the suffix class is onto. For finite sets, ontoness implies the domain is at least as large as the codomain. Thus $|Q| \geq |\mathcal{C}_{\text{Suffix}}(L)$.

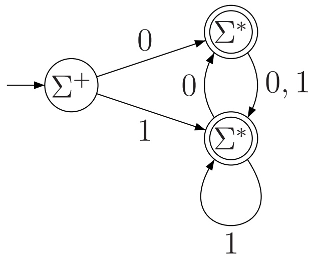
isomorphic to each other.[3] We say a DFA M is *unique (upto isomorphism)* if M is the only DFA with that number of states, ignoring other DFAs that are isomorphic to M.

The Myhill-Nerode theorem claims that M_{Suffix} is unique upto isomophism. This can be proven by showing that any other DFA with $|\mathcal{C}_{\text{Suffix}}(L)|$ states is isomorphic to M_{Suffix}, though we will not show that here. Intuitively, recall that each DFA state corresponds to some suffix language in $\mathcal{C}_{\text{Suffix}}(L)$. A unqiueness proof shows that M_{Suffix} is the only correct way to put transitions between the suffix language states. From another perspective, the 'uniqueness' of $\mathcal{C}_{\text{Suffix}}(L)$ forces the uniqueness of M_{Suffix}.

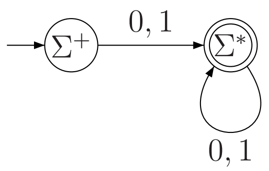
4.2 Finding the minimal DFA

Now a more pratical concern: how do we find the minimal DFA for a particular regular language L? We could build it if we knew $\mathcal{C}_{\text{Suffix}}(L)$, but determining the suffix class just from the language description seems like a difficult process. In fact, our main intuition about the suffix languages is from the states of a DFA, not from the language itself. Instead let's assume that we have a DFA M for L already. Our approach is to *optimize* M so that it becomes M_{Suffix}!

The DFA Minimization algorithm will determine which states have separate suffix languages, and thus which must be distinct states in M_{Suffix}. For example, consider the following DFA for the language $\mathcal{L}(\Sigma^+)$, where each state is labeled with the suffix language that it corresponds to:



The algorithm will determine that there is one state for the suffix language Σ^*, and thus the minimal version will collapse two states into one:



The algorithm works by *distinguishing* pairs of states that are different in the minimal DFA. That is, it attempts to find *distinguishing strings* - ones that are in one suffix language but not in the other - for each pair of states. The procedure follows:

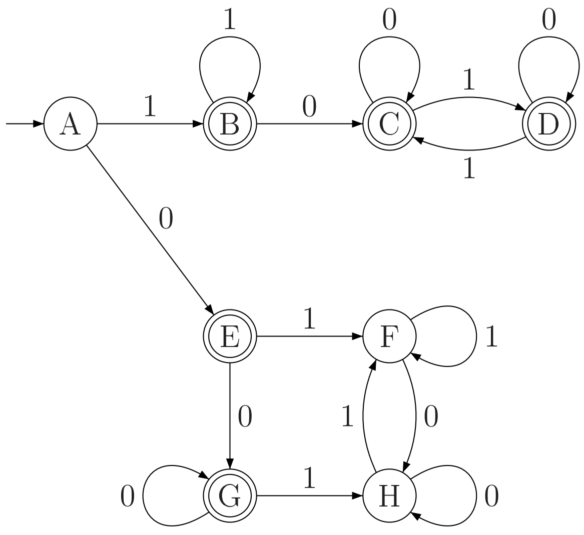
[3]This is the same graph isomorphism concept that you may have encountered in your intro discrete math class. Two isomorphic graphs have the same node and edge structure, though their nodes might have different names.

DFA Minimization:

1. Given input DFA $M = (Q, \Sigma, \delta, q_0, F)$, create a chart for each unordered pair of different states. All cells in chart are initially undistinguished.
2. For each $\{q, q'\}$ with $q \in F$, $q' \in Q \setminus F$ (i.e. exactly one is final)
   Mark the cell for $\{q, q'\}$ as distinguished
3. For each blank cell $\{q, q'\}$
   If there is an $a \in \Sigma$ such that $\{\delta(q, a), \delta(q', a)\}$ is distinguished
   then mark the cell for $\{q, q'\}$ as distinguished.
4. If any changes were made to the chart in Step 3, repeat from Step 3.
5. Extract the minimal DFA from the chart: undistinguished states are merged and the transitions are consistent with those in δ.

This method builds distinguishing strings recursively. In Step 2, the final and nonfinal states are distinguished, since ε is a distinguishing string: it's in the suffix languages of the final states, but not the nonfinal ones. In Step 3, new pairs become distinguished based on the existing ones.

Let's see an example minimization of the following DFA:



First we create the chart for each pair of distinct states:

B		-	-	-	-	-	-
C			-	-	-	-	-
D				-	-	-	-
E					-	-	-
F						-	-
G							-
H							
	A	B	C	D	E	F	G

Notice that $\{A, B\}$ and $\{B, A\}$ are the same unordered pairs, so half of the chart is unused. We distinguish the final and nonfinal states (Step 2):

B	x	-	-	-	-	-	-
C	x		-	-	-	-	-
D	x			-	-	-	-
E	x				-	-	-
F		x	x	x	x	-	-
G	x					x	-
H		x	x	x	x		x
	A	B	C	D	E	F	G

Next, we go through the blank cells and attempt to distinguish them (Step 3). Starting in the first column, notice that $\delta(A, 1) = B$ and $\delta(F, 1) = F$, and the $\{B, F\}$ cell is marked distinguished. Thus the $\{A, F\}$ cell becomes distinguished too:

B	x	-	-	-	-	-	-
C	x		-	-	-	-	-
D	x			-	-	-	-
E	x				-	-	-
F	**x**	x	x	x	x	-	-
G	x					x	-
H		x	x	x	x		x
	A	B	C	D	E	F	G

Similarly, $\{A, H\}$ is marked because of $\{E, H\}$ or $\{B, F\}$:

B	x	-	-	-	-	-	-
C	x		-	-	-	-	-
D	x			-	-	-	-
E	x				-	-	-
F	x	x	x	x	x	-	-
G	x					x	-
H	**x**	x	x	x	x		x
	A	B	C	D	E	F	G

We cannot mark $\{B, C\}$ or $\{B, D\}$, since none of the required state pairs are distinguished yet - though they might be in the future. The next state that can be marked is $\{B, E\}$ because $\{B, F\}$ is distinguished:

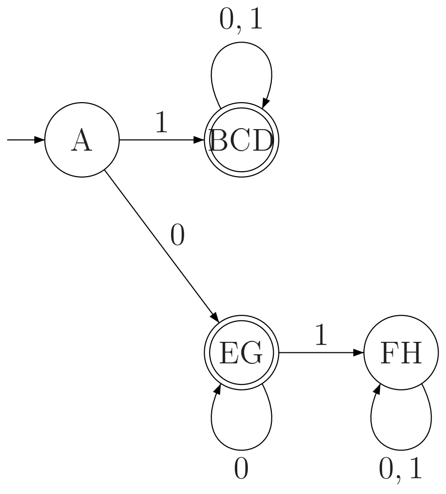
B	x	-	-	-	-	-	-
C	x		-	-	-	-	-
D	x			-	-	-	-
E	x	**x**			-	-	-
F	x	x	x	x	x	-	-
G	x					x	-
H	x	x	x	x	x		x
	A	B	C	D	E	F	G

And so on. Here's the chart after completing Step 3 (all additions in bold):

	A	B	C	D	E	F	G
B	x	-	-	-	-	-	-
C	x		-	-	-	-	-
D	x			-	-	-	-
E	x	**x**	**x**	**x**	-	-	-
F	**x**	x	x	x	x	-	-
G	x	**x**	**x**	**x**		x	-
H	**x**	x	x	x	x		x

We made changes to the chart in Step 3, so now we repeat the process on the remaining blank cells (Step 4). It's possible that some of the newly marked states will cause others to become marked as well. This does not happen in our example, so the second pass of Step 3 changes nothing. Thus the final chart is the same.

Now we can determine the minimal DFA from the chart. $\{B, C\}$, $\{B, D\}$, and $\{C, D\}$ are not distinguished, thus they are the same state in the minimal DFA. The same is true for $\{F, H\}$, and also for $\{E, G\}$. The transitions are the same as in the original DFA, noting that self loops might occur for a transition between a non-distinguished pair:



It turns out this was our original example from the previous lecture, which accepts $\mathcal{L}(1\Sigma^* \cup 0^+)$.