COSC 341
Theory of Computing
Lectures 7 and 8
Myhill-Nerode and its consequences

Stephen Cranefield

stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Keywords: Myhill-Nerode theorem, Moore's algorithm, Hopcroft's algorithm

A note about regular language closure properties (Tut. 5 update)

- Only <u>some</u> of the closure properties for regular languages can be proven constructively using NFAs (e.g. union, concatenation and Kleene-star).
- For complement, making accepting states non-accepting (and vice versa) works for DFAs, not NFAs.
- For string reversals, note that reversing transitions in a DFA will (in general) result in an NFA, but this is OK for proving this closure property.
- Intersection can be proven given union and complement using De Morgan's laws:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

- How can we show this constructively given NFAs $\mathbf{A_1}$ and $\mathbf{A_2}$ that accept L_1 and L_2?
  - Apply the subset construction to convert the NFAs to DFAs.
  - Switch accepting and non-accepting states in both DFAs.
  - Make these into NFAs in standard form.
  - Apply the NFA union construction.
  - Convert the result to a DFA via the subset construction.
  - Switch accepting and non-accepting states

Revision: State-equivalence

Let $\mathbf{A}$ be a DFA over Σ. Define a relation, $\sim_{\text{state}}$ on Σ^* called state equivalence by:

$w \sim_{\text{state}} v \iff$ the state reached in $\mathbf{A}$ by processing v is the same as that reached by processing w.

Revision: Suffix-equivalence

Given a language L over alphabet Σ, a distinguishing extension of two words $u, v \in \Sigma^*$ is any word $w \in \Sigma^*$ such that exactly one of uw and vw is in L.

Example: $L = \{a^n : n$ is even$\}$. For any k, a is a distinguishing extension of a^k and a^{k+1}.

> We define suffix equivalence (modulo L) as follows:
>
> $u \sim_{\text{suffix}} v \quad \Longleftrightarrow \quad$ there is no distinguishing extension for u and v

Alternatively, for any $w \in \Sigma^*$ define the suffix language of w modulo L:

$$\mathit{Suff}(w, L) = \{y \in \Sigma^* \ : \ wy \in L\}.$$

Then $u \sim_{\text{suffix}} v$ just means $\mathit{Suff}(u, L) = \mathit{Suff}(v, L)$.

The Myhill-Nerode theorem

Theorem

A language is regular if and only if its suffix-equivalence relation has only finitely many equivalence classes.

Revision: Uses of the Myhill-Nerode Theorem

1. Try to show that a language is regular by an exhaustive case analysis. Begin with ϵ and consider increasingly longer strings while trying to find distinguishing extensions until no more equivalence classes can be found.
2. Prove a language is not regular through logical analysis that shows there must be an infinite number of suffix-equivalence classes.

Example for case 2:

$L = \{a^n b^n : n \geq 0\}$.

- Given a^i and a^j for distinct i and j, consider the extension b^i.
- $a^i b^i \in L$ but $a^j b^i \notin L$.
- Thus b_i is a distinguishing extension and a^i and a^j are in different suffix-equivalence classes.

Part 1 of the theorem's proof

- Suppose that L is regular.
- There is a DFA, $\mathbf{A}$ that accepts it.
- The state-equivalence relation for $\mathbf{A}$ has only finitely many equivalence classes.
- If two words are state-equivalent then they are also suffix-equivalent.
- Therefore, the suffix-equivalence relation also has only finitely many classes.

Part 2 of the theorem's proof

- Suppose that the suffix-equivalence relation for L has only finitely many equivalence classes.
- Define a DFA, $\mathbf{A}$ whose states **are** (alternatively, are labelled by) the suffix-equivalence classes. The hypothesis is exactly that there are only finitely many of these.
  - Define the initial state as $[\epsilon]_{\sim_{\text{suffix}}}$
  - Define a state $[w]_{\sim_{\text{suffix}}}$ to be accepting if $w \in L$.
  - The transition on letter a from state $[w]_{\sim_{\text{suffix}}}$ is to $[wa]_{\sim_{\text{suffix}}}$.
- This DFA accepts (exactly) L.

Consequences of the proof

- If L is a regular language then there is a DFA accepting it such that the number of states of the DFA is equal to the number of equivalence classes of $\sim_{\text{suffix}}$.
- That's the smallest number of states possible since if two words are not suffix-equivalent, they cannot be state-equivalent.
- And in fact that minimal automaton is unique since the a-transition from a state corresponding to a particular suffix language must be to the state corresponding to all the words in that language beginning with a (after deleting the first character).
- If we are given *some* DFA can we construct the corresponding minimum one? This is called DFA minimisation.

DFA minimisation

The Myhill-Nerode Theorem shows there is a unique minimal DFA for any regular language. Suppose we have a DFA. Can we construct the unique minimal DFA in a systematic way?

Given a DFA $\mathbf{A}$ (we assume from now on that all states are reachable)

- What states are we *certain* correspond to different suffix-equivalence classes[1]?
- What starting partition does this give us that is *coarser* than the partition required for suffix-equivalence?
- How can we *refine* this partition?

[1] We can talk about suffix-equivalence for states, because all words that lead to that state are definitely suffix-equivalent

Minimisation idea (Moore's algorithm)

Given a DFA we want to find the equivalence relation on its states that corresponds to suffix-equivalence.

- Begin with an equivalence relation (or its partition) that we know is coarser than suffix-equivalence.
- Specifically, accepting states and non-accepting states are *not* suffix-equivalent (since the former accept ϵ and the latter don't).

Now loop based on the current partition.

- Within each class, see if we can tell things apart by looking at their transitions.
- If so, refine the partition to reflect this, and repeat.
- This terminates since a partition can't be properly refined infinitely often.

Are we done?

Why are we done?

Observation

The final partition of the states has the property that for any pair of states, x and y, in the same part and any letter, a, if

$$x \xrightarrow{a} x' \quad \textit{and} \quad y \xrightarrow{a} y'$$

then x' and y' are in the same part.

Suppose that, starting from x and y there is a distinguishing extension, i.e. for some word w we accept when x is followed by w but not when y is followed by w, or vice-versa.

Applying the observation above repeatedly this would mean that the states we reach on following w from x or from y would lie in the same part. But, one is accepting and the other isn't so they don't!

Moore's algorithm

- If we minimise a DFA by implementing the above in the most direct manner then we are performing Moore's algorithm
- In many practical contexts this is good enough.
- Examples.
- Worst-case complexity is $O(n^2|\Sigma|)$.
- At most n rounds are required.
- Each can be carried out in $O(n|\Sigma|)$ time if we maintain the states in a sorted order so that all states in the same (current) part are consecutive.

Hopcroft's algorithm

Let X and Y be subsets of the states and a a letter. Say that X *is split by* (a, Y) if there are some elements of X that go to Y under a and some that don't.

- Maintain both the current partition (coarser than $\sim_{\text{suffix}}$ modulo L) and a queue of sets from that partition called the *active splitters*.
- Both are initialised as the accepting/non-accepting partition.
- While the queue is not empty, remove its head B:
  - For each letter a and each current partition P that is split by (a, B):
    - Replace P in the current partitions by its split.
    - If P was an active splitter, replace it in the queue by both parts of the split.
    - If not, add just the smaller of the two new parts to the queue of active splitters.

Using the *partition refinement* data structure, Hopcroft's algorithm can be implemented with a run-time bound of $O(n|\Sigma| \log n)$.