IAS/PCMI Summer Session 2000
Clay Mathematics Undergraduate Program
Advanced Course on Computational Complexity

# Lecture 3: Algebra and Languages

David Mix Barrington and Alexis Maciel
July 19, 2000

## 1. Overview

In this lecture we go into more depth about the algebraic model of computation, viewing languages as inverse images under homomorphisms of subsets of finite monoids. Specifically:

- We prove that given any language we can construct a particular monoid by purely combinatorial means, and that the language is regular iff that monoid is finite.
- By a similar argument we prove the *Myhill-Nerode Theorem*, which lets us find the smallest possible finite automaton for a language, or prove that a language is not regular.
- We apply the Myhill-Nerode Theorem to show that constant-space machines, or two-way finite automata, can decide only regular languages.
- Finally we define *aperiodic monoids* and prove that any first-order definable language is recognized by an aperiodic monoid. Though we won't prove this completely, aperiodics define the algebraic complexity class that is equal to the first-order definable or star-free regular languages.

## 2. Syntactic Congruences and Monoids

The *syntactic congruence* of a language $L \subseteq \Sigma^*$ is a particular equivalence relation on the strings in $\Sigma^*$. Two strings will be considered to be equivalent if they behave the same with respect to membership in $L$, particularly if no *experiment* of a certain kind can distinguish between them.

Specifically, we say that $u \equiv_L v$ is true iff for any two strings $x$ and $y$, the strings $xuy$ and $xvy$ are either both in $L$ or both not in $L$. That is, we cannot distinguish $u$ from $v$ by placing a string before each, a string after each, and testing the results for membership in $L$. The equivalence classes of this relation (you'll prove in the exercises that it's an equivalence relation) form the *syntactic monoid* of $L$. To make the set of classes into a monoid, of course, we must define an operation on them and check the monoid properties.

If $u$ and $v$ are strings, we write their equivalence classes as $[u]$ and $[v]$. The product of $[u]$ and $[v]$ in the monoid is defined to be the class $[uv]$ containing the concatenation $uv$. For this definition to be valid, we must show that it actually defines a single class. That is, we need to know that if $u \equiv_L u'$ and $v \equiv_L v'$, then $uv \equiv_L u'v'$. (That is, different choices of $u$ or $v$ within the same class always lead to the same product class.) Why is this true? Well, for arbitrary $x$ and $y$, $xuvy$ is in $L$ iff $xu'vy$ is in $L$ because of the equivalence of $u$ and $u'$. Similarly, $xu'vy$ is in $L$ iff $xu'v'y$ is in $L$ because of the equivalence of $v$ and $v'$.

The monoid properties are easy to check once we've shown this. The class $[\lambda]$ is the identity because $\lambda$ is the identity for string concatenation, and the monoid operation is associative because string concatenation is associative. We've also shown that the function taking any string $u$ to its equivalence class $[u]$ is a monoid homomorphism, explicitly showing that $L$ is recognizable *if* the syntactic monoid is finite. This is one half of:

**Theorem 1** *A language $L$ is recognizable (by a homomorphism into a finite monoid) iff its syntactic monoid is finite.*

**Proof** We have only half the proof left — we must show that an arbitrary recognizable language has a finite syntactic monoid. Let $L$ be the inverse image of $X \subseteq N$ under some monoid homomorphism $\phi$ from $\Sigma^*$ to some finite monoid $N$. The inverse images of elements of $N$ under $\phi$ form a partition of $\Sigma^*$ into finitely many classes. I claim that the classes of the syntactic congruence are unions of these classes (and thus that there are only finitely many congruence classes). If two strings $u$ and $v$ satisfy $\phi(u) = \phi(v)$, consider arbitrary strings $x$ and $y$ and note that because $\phi$ is a homomorphism, $\phi(xuy) = \phi(xvy)$. If this element of $N$ is also in $X$, both $xuy$ and $xvy$ are in $L$, otherwise neither is in $L$. □

The proof above actually gives us something more — a homomorphism $\psi$ from $N$ to the syntactic monoid $M$, where $\psi(n) = [u]$ for any element $u$ such that $\phi(u) = n$.

(Actually this homomorphism is only defined on the subset of $N$ that is the image of $\Sigma^*$ under $\phi$.) A key notion in algebraic automata theory is that of one monoid *dividing* another. Here we can say that $M$ divides $N$ because it is the homomorphic image of a subset of $N$. Division places a partial order on the set of all monoids, and we've shown that the syntactic monoid is *minimal* in this order among all monoids that recognize $L$.

# 3. The Myhill-Nerode Theorem

It's a natural question to ask about the "best possible" decision algorithm for a language of a certain kind. We've found the smallest possible monoid that recognizes $L$, namely the syntactic monoid. But there is a possible way to do "better", in one sense. If we think of the monoid as a set of transformations of a finite set, we could ask to minimize the size of the set rather than the size of the transformations. If we translate from the monoid language back into the older language of deterministic finite automata, then this set corresponds to the *states* of a DFA, and we would then be finding a DFA for the language with the smallest possible number of states.

The Myhill-Nerode theorem uses reasoning like that above to characterize this minimal set as the classes of an equivalence relation on strings. The *Myhill-Nerode congruence* on $\Sigma^*$ for a language $L \subseteq \Sigma^*$ is as follows. If $u$ and $v$ are strings, we now say that $u \sim_L v$ if for any string $x$, $ux$ and $vx$ are either both in $L$ or both not in $L$. That is, a one-sided version of the experiment in the last section cannot distinguish $u$ from $v$. Note that if $u \equiv_L v$, it follows that $u \sim_L v$, but not conversely. Also note that the classes of the MN congruence do not form a monoid, but that elements of the monoid can be thought of as acting as transformations on the MN classes. (If $[x]$ is an MN class and $[u]$ is a monoid element, the result $[xu]$ does not depend on which $x$ or $u$ in the class is chosen, and this can be defined to be $\delta([x],[u])$.)

**Theorem 2** *(Myhill-Nerode) A language is recognizable iff its MN congruence has only finitely many classes. It has a DFA with at most $k$ states iff the MN congruence has at most $k$ classes. (See Basic Lecture 3 for a definition of a DFA.)*

**Proof** If a language is recognizable, we have proved it has only finitely many classes in its syntactic congruence, and the MN classes are unions of these. Conversely, the set of all transformations on the finite set of MN classes forms a finite monoid, and it is not hard to show that the natural mapping from strings to transformations is a homomorphism that recognizes $L$.

If $L$ has a DFA $M$ with at most $k$ states, we note that if both $u$ and $v$ take $M$ from its initial state to some state $s$, we know that $u \sim_L v$. This is because the strings $ux$ and $vx$ necessarily take $M$ to the same state (wherever $x$ takes $M$ from $s$), and this state is either a final state of $M$ or it isn't. The MN classes are thus unions of the at most $k$ classes defined from states in this way.

If the MN congruence has at most $k$ states, we can define a DFA with the same number of states that decides $L$. We have a state for each class, and each letter $a$ takes the state $[x]$ to the state $[xa]$. It should be clear that if $x \sim_L y$, then also $xa \sim_L ya$, so this mapping is well-defined. When the DFA has read an entire word $w$, it is in state $[w]$, and any class of the MN congruence is either entirely within $L$ or disjoint from $L$. So the DFA decides $L$ with a proper choice of final states. $\square$

A common use of this theorem is to show that a language is *not* recognizable (and thus not regular) by showing that it has infinitely many MN classes. For example, the language $EQ = \{a^n b^n : n \geq 0\}$ is not regular because for any distinct numbers $i$ and $j$, the strings $a^i$ and $a^j$ are not MN-equivalent. (To prove this, note that $a^i b^i$ is in $EQ$ and $a^j b^i$ is not.) Thus each $a^i$ is in a separate class, and there are infinitely many such classes.

This is conceptually simpler than the proofs of non-regularity usually presented, using a *pumping lemma*. One good reason to present pumping lemmas for regular languages is to prepare students for the more complicated pumping lemmas used to show languages to be not context-free, but we have no need for this here.

# 4. Two-Way Automata and Regular Languages

Consider a machine with a read-only input tape (on which it can move right or left one space at a time) and a worktape of $O(1)$ bits. We can make an equivalent machine with no worktape at all, storing the contents of the former worktape in the state of the machine. Thus an arbitrary DSPACE(1) machine can be thought of as a *two-way deterministic finite automaton*. Since an ordinary DFA is a special case of a 2WDFA, 2WDFA's are capable of deciding all regular/recognizable languages. Are they any more powerful? After all, our lower bound proofs for regular language seem to rely on the fact that once an ordinary DFA has passed over some input, it cannot look at it again. This is simply not true of a 2WDFA. But we will now see that any 2WDFA recognizes a regular language, and thus is equivalent to an ordinary DFA (though in general one with vastly more states. The Myhill-Nerode Theorem will simplify this proof enormously.

First let's notice that the semantics of a 2WDFA can be more complicated than those of an ordinary DFA. If we start at the left end of a word in the initial state, the ordinary DFA is guaranteed to leave the word at the right end in some state, so we know it will either accept or reject the word based on this state. A 2WDFA has two other possible behaviors — it could leave the *left* end of the word, or it could enter a loop and never leave the word. If $M$ is a 2WDFA, we define $L(M)$ to be the set of strings that are accepted by $M$, so that the strings neither accepted nor rejected are *not* considered to be in $L(M)$.

Fixing a 2WDFA $M$ with $k$ states $\{q_1, \ldots, q_k\}$, we'll define an equivalence relation $\simeq$ on strings such that if $x \simeq y$, $x$ and $y$ are also in the same MN class for $L(M)$. Since it will be clear that $\simeq$ has finitely many equivalence classes, it will then follow immediately that $L(M)$ is recognizable.

We will define $k+1$ functions, each from $\Sigma^*$ to the set $\{q_1, \ldots, q_k, d\}$ where $d$ will represent "death" (moving off the left of the tape or entering a loop). Given an arbitrary $w \in \Sigma^*$:

- $f_0(w)$ is the state in which $M$ leaves $w$ to the right, after starting on the left of $w$ in the initial state. If $M$ never leaves $w$ on the right in this case, $f_0(w) = d$.
- For each state $q_i$, $f_i(w)$ is the state in which $M$ leaves $w$ to the right, after starting on the *right* of $w$ in state $q_i$. If $M$ never leaves $w$ on the right in this case, $f_i(w) = d$.

We define $u \simeq v$ to be true iff $f_i(u) = f_i(v)$ for all $i$ from 0 through $k$. It remains to prove that if $u \simeq v$, then $u$ and $v$ are MN-equivalent. So let $x$ be an arbitrary string, assume $u \simeq v$, and we will show that $ux$ and $vx$ are either both in $L$ or both not in $L$.

Consider the computations of $M$ on the two strings $ux$ and $vx$, and call the left end of $x$ the "midpoint" of each of these strings (though we know nothing about the relative lengths of $u$, $v$, and $x$). Each computation starts at the left end of its string in the initial state. If $f_0(u) = f_0(v) = d$, both computations die and neither $ux$ nor $vx$ is in $L$. Otherwise both reach the midpoint in the same state $f_0(u) = f_0(v)$. Now both computations proceed identically in $x$, until either both accept, both reject, both enter a loop, or both go back over the midpoint. We are done in all of these cases except the last. In that case both cross the midpoint in some state $q_i$, and then either both die or both return to the midpoint in the same state $f_i(u) = f_i(v)$. The two computations continue to mirror each other in this way, either because they are traversing the identical strings on the right (copies of $x$) or the sufficiently similar strings on the left ($u$ and $v$) of the midpoint. Eventually either both die, both accept, or both reject. (If, for example, both computations cross the midpoint more than once in the same state, both will loop forever.) We could be more formal about this by using induction on the number of crossings of the midpoint, but the idea should be clear — the end results (if any) of the two computations will be the same, and this the two strings are either both in $L$ or both not in $L$.

# 5. Aperiodic Monoids

We have shown that the classes of regular and of star-free languages are quite robust, each having several definitions in different models. (For many more definitions of the star-free languages, consult *Counter-Free Automata* by McNaughton and Papert.) A natural question to ask is whether the star-free/first-order languages can be defined in the algebraic model. They can, and here we will state the characterization and prove one half of it. The other half would take us too far afield into algebra.

A finite monoid $M$ is defined to be *aperiodic* if there is some number $t$ such that for every element $a$ of $M$, $a^{t+1} = a^t$. (Here the exponentiation denotes repeated use of the monoid's "multiplication" operation, as usual.) In any finite monoid, the powers of $a$ cannot all be distinct, so we must have $a^{t+q} = a^t$ at some point. It is not hard to show that for some $t$ and some $q$ this equation holds for all elements $a$.

A simple aperiodic monoid is the *threshold monoid* $T_n$, consisting of the numbers $\{0, 1, \ldots, n-1\}$ under addition, with the proviso that all numbers greater than $n-1$ are considered to be equal to $n-1$. A *boolean algebra* forms an aperiodic monoid under either its AND or OR operations — one example is the $2^n$ boolean vectors of length $n$ under bitwise AND or OR.

The characterization is that a language $L$ is star-free iff it is *recognized by an aperiodic monoid*. That is, there must be an aperiodic monoid $A$, a homomorphism $\phi$ from $\Sigma^*$ to $A$, and a subset $X$ of $A$ such that $L = \phi^{-1}(X)$. For example, the set of all strings $w$ containing at least three $a$'s is recognizable by the aperiodic monoid $T_4$, with $\phi$ taking $a$ to 1 and all other letters to 0, and $X = \{3\}$.

We can use EF games to show that any first-order definable language is recognized by an aperiodic. Recall that for any $k$, the $k$-move EF game divides $\Sigma^*$ into a large number of equivalence classes, where two strings are equivalent iff Delilah wins the $k$-move game on them. We showed in Advanced Lecture 2 that there is a well-defined concatenation operation on EF classes and that the classes form a finite monoid $EF_k$ under this operation. The homomorphism taking each string to its class can be made to recognize *any* language definable by a $k$-quantifier first-order formula, given a suitable choice of a subset of $EF_k$. It remains for us to show that $EF_k$ itself is aperiodic. The correct value of $t$ turns out to be $2^k - 1$.

**Lemma 3** *For any $k$ and any $w$, Delilah wins the $k$-move EF game on the strings $w^{2^k-1}$ and $w^{2^k}$.*

**Proof** We merely adapt the proof in Advanced Lecture two that Delilah wins the $k$-move game on $a^{2^k-1}$ and $a^{2^k}$, where $a$ is a single letter. We use induction on $k$ and note that for $k = 0$ there is nothing to prove since all strings are equivalent. For arbitrary $k$, we must define Delilah's response to any opening move of Samson's, and show that she wins the resulting $k - 1$ move game. Samson's first move is into some copy of $w$. Delilah moves to the corresponding character of some copy of $w$ in the other string. She chooses her copy so that the number of copies of $w$ to the left of hers is equal to the number to the right of Samson's, *unless* both of these numbers are at least $2^{k-1} - 1$. Similarly she chooses it so that the number of copies of $w$ on the right is the same as for Samson's, unless both of these numbers are at least $2^{k-1} - 1$. Since each string has at least $2^k - 1$ copies, only one of these conditions constrains Delilah and she can follow this strategy. By induction, she wins the $k - 1$-move game on either the left or the right, so at the end of $k - 1$ more moves of Samson (split between the left and right as he likes) she will have winning positions in both games and thus overall. □

Proving that all languages recognized by an aperiodic are first-order definable would require a better characterization of what kinds of aperiodic monoids are possible. Such a thing exists, but is beyond the scope of these lectures. Just as there is the Jordan-Holder structure theorem for finite groups, showing how any group can be made up from simple groups by a certain product operations, the *Krohn-Rhodes Theorem* gives a similar structure property for all finite monoids. (The basic building block for aperiodics is the monoid $T_2$ defined above.) We'll refer to the KR theorem in passing once or twice later on, but not prove it or even formally state it.

# 6. Exercises

1. Prove that the syntactic congruence and the Myhill-Nerode congruence are each equivalence relations.

2. Find the syntactic monoid and the Myhill-Nerode classes of the language $\Sigma^* ab \Sigma^*$.

3. Prove that the division relation on monoids is a partial order.

4. Prove that the set of unary strings of prime length is not regular.

5. Let $M$ be a 2WDFA with states $p$ (initial), $q$ (only final), and $r$. Define the transition function by $\delta(p, a) = (q, R)$, $\delta(p, b) = (r, L)$, $\delta(q, a) = (q, R)$, $\delta(q, b) = (r, L)$, $\delta(r, a) = (p, R)$, and $\delta(r, b) = (q, L)$. Find the $\simeq$ classes for $M$ and thus the language $L(M)$. (Hint: there are 256 possible values for the sequence of functions $f_0$, $f_p$, $f_q$, $f_r$, but presumably most of them never occur for any string.)

6. Prove that any finite monoid satisfies $\forall a : a^{t+q} = a^t$ for some number $t$ and some number $q > 0$. Prove that the monoid is a group iff $t$ can be taken to be zero. Are any monoids both groups and aperiodics?

7. Prove that if $\phi : A \to B$ is a monoid homomorphism that is onto, and $A$ is aperiodic, then $B$ is aperiodic. Argue that if $B$ divides $A$ and $A$ is aperiodic, then $B$ is aperiodic.