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

Lecture 2: Connecting the Three Models

David Mix Barrington and Alexis Maciel
July 18, 2000

1. Overview

Now that we have defined our three new models of computation (algebra, regular expressions, and first-order logic) we can explore connections among them. We will see that two robust complexity classes emerge from more than one of these models:

- The *regular* or *recognizable* languages, definable as those that have regular expressions *or* as those that are inverse images of a subset of a finite monoid under a homomorphism, and
- The *star-free* or *first-order definable* languages.

The two theorems we will help demonstrate that these are mathematically interesting classes of languages. Also, they provide a means to use upper and lower bound techniques designed for one model to get results in one of the other two models.

2. Kleene's Theorem

The fact that finite-state machines decide exactly the regular languages is often proved in a discrete mathematics or theory of computation class. We'll do the high points of this proof here in our algebraic setting, deferring some of the details to the exercises.

As a technical artifact for use in the proof, we'll define a *generalized nondeterministic finite automaton* or *GNFA*. (Nondeterminism as a general phenomenon will be explored in Basic lecture #3 — for the moment let's just consider GNFA's to be an *ad hoc* model.) A GNFA is a directed finite graph where each edge is labelled by a regular expression. There is a single node called the *start state* and another different node called the *final state*. Edges may go anywhere *except* into the start state or out of the final state (so that any other state may have a loop from itself to itself).

A GNFA M defines a language $L(M)$ in a way that is not immediately obvious. A word w is in $L(M)$ *if* it is *possible* to find (a) a path in M from the start state to the final state, consisting of edges $e_1, \ldots, e_k$, and (b) words $u_1, \ldots, u_k$, such that $w = u_1 \cdots u_k$ and each word u_i is in the regular language defined by the label on e_i. It is *not* obvious, given w and M, how to decide whether $w \in L(M)$. (Note that there is at least a Turing-computable but time-consuming way to do it — list all possible paths and consider all possible divisions of w into the right number of strings. Even this, of course, requires an argument to get an upper bound on the length of the longest path we need to consider. The proofs in this lecture will, among other things, give us a fairly efficient way to decide whether $w \in L(M)$.)

A *two-state* GNFA has at most one edge, since going into the start state or out of the final state is forbidden. Thus the language of a two-state GNFA is equal to the language of the regular expression on that edge (or to $\emptyset$, if there is no such edge). This suggests an algorithm to convert GNFA's into regular expressions with the same language (and thus prove that GNFA's define only regular languages). We need to take a GNFA with more than two states and *eliminate* one of the middle states to produce a new GNFA with the same language and one fewer state. Repeated applications of this process will get us to two states, and thus to our regular expression.

The procedure to eliminate a state is fairly simple and relies on the fact that simple combinations of regular expressions are also regular expressions. Suppose that the state s is to be eliminated, and that s has a self-loop labelled with the regular expression β. If s occurred on a path from the start state to the final state, that path must have entered s from some other state p, looped at s zero or more times, and then gone from s to some state q (which might equal p). If α is the label on the edge from p to s, and γ the label on the edge from s to q, we can include all possible trips through s of this kind by a single edge from p to q, labelled $\alpha\beta^*\gamma$. The reader may verify that if we delete s but add such edges for all possible choices of p and q, we get exactly the same possible paths from the start state to the final state and thus exactly the same language. (If we "add an edge" to a place where an edge already exists, this means we union the new regular expression with the old one. In our example above, if there was already an edge from p to q labelled δ, our new GNFA would have an edge from p to q labelled $\delta \cup (\alpha\beta^*\gamma)$.)

We've now shown that every GNFA language is regular. We'll complete Kleene's Theorem by showing that every recognizable language has a GNFA, and that every regular language is recognizable. The first step is easy. Consider a language L which is $\phi^{-1}(X)$ for some homomorphism $\phi : \Sigma^* \to M$ and some set $X \subseteq M$. Our GNFA's nodes will consist of one node for each element of M, plus a start and a final state. For every element x of M and every letter a, we make an edge of the GNFA from x to $x\phi(a)$ labelled by a. Then we make edges from the start state to e (the identity of the monoid) and from each element of X to the final state. These edges are called *λ-moves* because they are labelled by the regular expression "λ" or $\emptyset^*$. The language of this regular expression contains exactly one string, the empty string λ. There is thus a path through the GNFA iff we can go from the start state to e, through a letter-move for each letter of w until we reach the node corresponding to $\phi(w)$, and then to the final state on a λ-move. This last is possible iff $\phi(w) \in X$, that is, iff $w \in L$.

The final transformation from regular to recognizable languages will take place in two phases. Given a regular expression, we first construct a GNFA by induction on the structure of the regular expression, whose language is that of the expression:

- The GNFA for $\emptyset$ has a start state, a final state, and no edges.
- The GNFA for a has a start state, a final state, and one edge labelled a from the start to the final state.
- If α and β have GNFA's M and N respectively, we construct a GNFA for $\alpha \cup \beta$ by taking a copy of M and a copy of N, merging the two start states, and merging the two final states.
- Similarly, we make a GNFA for $\alpha\beta$ by taking the two copies and merging the final state of M with the start state of N. The new GNFA's start state is the start state of M and its final state is the final state of N.
- If α has a GNFA M, we make a GNFA for α^* by taking a copy of M and adding two new states (a new start state s and a new final state f) and four λ-moves. These go from s to the old start state, from the old start state to the old final state and vice versa, and from the old final state to f.

The reader may verify, with careful reference to the given properties of GNFA's, that this construction is correct. The more complicated construction for the star operation is necessary so that the constructed GNFA will have no edges into its start state or out of its final state. Note also that the GNFA constructed has all its edges labelled either by letters or by λ.

The next step is to take the resulting GNFA and kill its λ-moves, to get an equivalent GNFA with the same nodes where all the edges are labelled by letters (or unions of letters). Unfortunately (see the exercises) we cannot do this given our exact definition of a GNFA, but (also see the exercises) we can do it if we allow the start state to also be a second final state. We will finish the proof of Kleene's Theorem by showing that if N is any GNFA whose labels are letters or unions of letters (any NFA, using the terminology of Basic Lecture 3), then $L(N)$ is recognizable.

Our monoid M will be the set of directed bipartite graphs whose left and right edge sets each correspond to the nodes of N. An edge from the left node corresponding to x to the right node corresponding to y will correspond to a possible path from x to y in N. The identity of M will be the graph which has only edges from x to x for each node x of N. The graph $\phi(a)$, for $a \in \Sigma$, will have an edge from x to y iff there is an edge with label including a from x to y in N. The operation of M is as follows. If G and H are graphs in M, make a new graph I by merging the right node set of G with the left node set of H. Then make a new graph GH that has an edge from x to y iff there are one or more two-step paths from the left node x to the right node y in I.

Now if w is a word in Σ^*, $\phi(w)$ will be a graph that has an edge from x to y iff there is a path from x to y in N whose labels consist of the letters of w in order. We thus choose X to be the set of graphs in M that have an edge, from the left node corresponding to the start state of N, to a right node corresponding to a final state of N.

Of course this construction accomplishes the same task as the "subset construction" of Basic Lecture 3, which converts an NFA to a DFA. The reader may wish to investigate the similarities between the two constructions.

3. From Star-Free to First-Order

We now show that the star-free regular and first-order definable languages are the same, proving each direction separately in the next two sections. Since we have inductive definitions of both star-free regular expressions and of first-order formulas (and their languages of marked words), we can use induction to construct an expression for every formula and a formula for every expression. In each case *most* of the cases of the construction will be straightforward, but some will offer technical difficulties.

For each star-free regular expression, we want a first-order sentence defining its language. For the empty set we may say $\exists x : (x \neq x)$, and for the language $\{a\}$ we may say $\exists x : \forall y : I_a(x) \wedge (x = y)$. The boolean operations of union and complementation correspond direction to the boolean operations $\vee$ and $\neg$ on sentences. We are thus left with the concatenation operator on languages: given star-free languages S and T with corresponding sentences Φ and Ψ, we need to construct a sentence for the language ST.

The main trick is as follows. Given any formula Φ and any variable x not occurring in Φ, we want to construct new formulas $\Phi_{<x}$ and $\Phi_{\geq x}$ that each have x as a free variable. If w is a marked word with an x-mark in position i, then $\Phi_{<x}$ will be true iff the subword to the left of i satisfies Φ. Similarly $\Phi_{\geq x}$ will be true iff the subword consisting of the letter in position i and everything to its right satisfies Φ. Once these are constructed, an arbitrary word will be in ST iff it satisfies the sentence

$$\exists x : (\Phi_{<x} \wedge \Psi_{\geq x}).$$

(This actually isn't quite true. It is right if the empty string does not itself satisfy Ψ, but if it does we must OR the sentence above with Φ (to cover the possible parsing of w as $w\lambda$). In either case the sentence for ST exists.)

Constructing $\Phi_{<x}$ and $\Phi_{\geq x}$ by induction on Φ is straightforward. The atomic formulas are unchanged, and the boolean operators simply carry over (for example, $(\Phi \wedge \Psi)_{<x}$ is just $\Phi_{<x} \wedge \Psi_{<x}$). The only non-trivial part is that we must add an additional clause whenever a quantifier is used. For example, if Φ is $\exists y : \Psi$, then $\Phi_{<x}$ is $\exists y : (y < x) \wedge \Psi$. If Φ is $\forall y : \Psi$, then $\Phi_{<x}$ is $\forall y : (y < x) \rightarrow \Psi$. Of course the reader may check that this construction is correct.

4. From First-Order to Star-Free

We now have to prove that any language of marked words defined by a first-order formula is star-free regular. Before we begin, we need to establish what a "star-free language of marked words" means. We can represent marked words as strings if we use an extended alphabet, with letters a_S for every ordinary letter a and every set S of marks. If x is a variable, we define the alphabet Σ_x to be all marked letters where x is *one of* the marks. (The set of all a's with such marks will be denoted a_x.) Similarly, if X is a set of marks, Σ_X is the set of all marked letters where all the marks come from X.

Now, of course, not all strings of marked letters form valid marked words. Fix X as a set of marks. A valid marked word with marks from X must have *exactly one* copy of each mark in X. Fortunately, this condition is star-free. For each mark x, the set of marked words with exactly one x-mark is the star-free language $\Sigma^*_{X-\{x\}}\Sigma_x\Sigma^*_{X-\{x\}}$. The language V_X of all valid marked words with mark set X is the intersection of these languages for all $x \in X$. (Since X is finite, this is a finite intersection and the result of it is guaranteed to be star-free regular.)

Now we can begin our inductive definition of a language for each formula. (We fix an alphabet Σ for the whole construction.) For any set of marks X, we first specify a language of the valid marked words satisfying each atomic formula:

- The language for "$I_a(x)$" is $V_X \cap (\Sigma_X^* a_x \Sigma_X^*)$.
- The language for "$x = y$" is $V_X \cap (\Sigma_X^*(\Sigma_x \cap \Sigma_y)\Sigma_X^*)$.
- The language for "$x < y$" is $V_X \cap (\Sigma_X^* \Sigma_x \Sigma_X^* \Sigma_y \Sigma_x^*)$

Once again, boolean operations on formulas correspond to boolean operations on languages. The only difficult case of the induction is the case of quantifiers (we can restrict attention to existential quantifiers). Suppose Φ is the formula $\exists x : \Psi$, where x is free in Ψ. When does a marked word W satisfy Φ? Exactly when it is possible to write W as Ua_SV, where U and V are marked words with no x-mark, S is a set of marks (possibly empty) not containing x, and the marked word $Ua_{S\cup\{x\}}V$ satisfies Ψ.

Fortunately, we can divide all the ways that this might happen into a large but *finite* set of cases. If X is a set of marks and k is a number, the set V_X is divided into *equivalence classes* by a relation called *k-equivalence*: two marked words are k-equivalent iff they satisfy *exactly the same* first-order formulas with k or fewer quantifiers and free variables exactly those in X. We will need two lemmas about k-equivalence:

Lemma 1 *For every X and k there are only finitely many k-equivalence classes in V_X.*

Proof We use induction on k. For $k = 0$, note that there are only finitely many distinct atomic formulas (once X is fixed) and thus only finitely many different boolean combinations of these. (There are infinitely many formulas, but since there are only finitely many boolean combinations all but finitely many of the formulas have a shorter equivalent form.) If two marked words satisfy exactly the same subset of these finitely many formulas, they are 0-equivalent.

For the inductive case, we assume that there are finitely many different formulas with k quantifiers. For each such formula Ψ, we can make finitely many formulas of the form $\exists x : \Psi$. The resulting finite set of "primitive" $k+1$-quantifier formulas may then produce only finitely many more formulas by boolean combination. This completes the induction and thus the proof. $\square$

Lemma 2 *If S and T are k-equivalence classes (with disjoint sets of marks), so is their concatenation ST.*

Proof Recall the Ehrenfeucht-Fraïssé games from an exercise in Advanced Lecture 1. Two marked words U and V are k-equivalent iff the "duplicating player", Delilah, wins the k-move EF game on U and V. Consider two marked words U and V in the language ST. It must be possible to write U as U_1U_2 and V as V_1V_2 such that U_1 and V_1 are in S and U_2 and V_2 are in T. Delilah finds such words as part of her strategy. She will respond to each move of Samson by moving in the corresponding part of the other word — for example, if Samson moves in U_1 she replies by making a mark in V_1, and similarly for the other three cases. Delilah chooses her move by following the winning strategies she is *assumed* to have for the k-move games on the pairs of strings $\{U_1, V_1\}$ and $\{U_2, V_2\}$. These strategies exist because each of these two pairs of strings are from the same k-equivalence class.

At the end of the game, Delilah is assured that all atomic formulas have the same value in the two marked versions of U and V. For letter and equality formulas this follows from the fact that she has won the two subgames. For order formulas such as $x < y$, this is assured either by winning a subgame or by the fact that x is marked in the left subgame and y in the right one. Since Delilah has a winning strategy on U and V, these marked words are k-equivalent. □

We may now finish our construction of a star-free regular expression for each formula. Recall that Φ is the formula $\exists x : \Psi$, where Ψ has k quantifiers and has x as a free variable. For each possible pair of k-equivalence classes S and T, and each possible marked letter a_Y with $x \in Y$, the language Sa_YT is an equivalence class (if it is a set of valid marked words at all). Thus either all the marked words in such as language satisfy Ψ or none of them do. There are only finitely many of these combinations, so we can take all of those that satisfy Ψ and union together the corresponding languages $Sa_{Y-\{x\}}T$ to get the language of words satisfying Φ. Since this union is finite, we can get a star-free regular expression for the language of Φ.

5. Coming Attractions

We now have multiple characterizations of both the star-free regular languages and all the regular languages. One natural question we have not yet answered is whether it is possible that these two classes of languages are *equal*. We saw that a language such as $(ab)^*$, where the star might appear to be essential, has an equivalent star-free form. Can all stars be eliminated? If the answer is yes, we could prove it by showing how to do it. If the answer is no, we can only prove it by a *lower bound* argument, showing that *no* star-free regular expression can define some particular regular language.

As one could perhaps guess, the answer is no. The simplest example of a regular language that is not star-free is $(aa)^*$, the set of all strings of a's of even length. One way to prove this is to characterize the set of all star-free regular languages that are *unary* (have only one kind of letter) as those sets of lengths that are *finite* or *cofinite*. This is not hard to do by induction on star-free regular expressions.

But another proof follows from our equivalence of star-free regular and first-order definable languages. We can show that for each k, all sufficiently long strings of a's are k-equivalent to each other. This is not hard to do using EF games, by induction on k. For $k = 0$, all strings with no marks are 0-equivalent and Delilah wins the 0-move game on any two strings. For $k = 1$ Samson can win if one string is empty and the other isn't, but on any two nonempty strings of a's any move for Delilah is a winning one. The case of $k = 2$ illustrates the general principle. When Samson moves on one string, he divides that string into the piece to the left of his mark and the piece to the right. Delilah must then divide the other string into an equivalent left piece and an equivalent right piece. For example, Samson wins the game on strings aa and aaa. He could move to the middle place of aaa and Delilah cannot make both the left and right pieces of aa nonempty, so she loses. But if each string has three or more a's, Delilah wins by moving to the right or left end if Samson does, and otherwise moving somewhere in the middle.

In general Delilah wins the k-move game if both strings have at least $2^k - 1$ a's, because she can move so that the left and right pairs of substrings are either equal or both have at least $2^{k-1} - 1$ a's. Then the inductive hypothesis says that she can win the $k - 1$-move game on either pair of substrings.

The other natural question remaining from our analysis is whether we can characterize the star-free regular languages in the algebraic model. This is possible, though we will be able to prove only one direction of the characterization in these lectures. One can also ask whether the model of first-order expressibility can be *extended* to include the regular languages. This is also possible, though the answer would take us somewhat deeper into monoid theory that we're willing to go here. What we will do, particularly in Lectures 4 and 5, is consider extensions of these models that characterize complexity classes originally defined in terms of machines and circuits.

6. Exercises

1. Find a star-free regular expression equivalent to λ.

2. Show that if A is any subset of Σ, A^* is star-free (this was used extensively above).

3. Let EE be the language over $\Sigma = \{a, b\}$ consisting of all strings with both an even number of a's and an even number of b's. Find a regular expression whose language is EE. (Hint: make a GNFA and use the given construction.)

4. Prove that there are regular languages for which there is no GNFA such that all edges are labelled by letters or unions of letters.

5. Given a GNFA M labelled with letters, unions of letters, and λ, construct an equivalent GNFA N labelled only by letters or unions of letters *except* that the start state *may* also be a second final state. (Hint: Find all possible paths in M consisting of zero or more λ-moves, a letter-move, and zero or more λ-moves and make a letter-move in N corresponding to each. Show that any path in M corresponding to a *non-empty* string w may be divided into one or more of these paths and thus corresponds to a path in N. Make the start state of N final iff there is a path of λ-moves in M from the start to the final state. Show that this does *not* allow any paths from start to final states in N that do not correspond to equivalent paths in M.)

6. Let $\Sigma = \{a, b\}$. Using EF games, find the k-equivalence classes of strings in Σ^* for k as large as possible. For $k = 0$ all strings are 0-equivalent. There are four 1-equivalence classes. If my count is right, there are 97 2-equivalence classes. Can you get an upper bound on the number of 3-equivalence classes?