Inexpressibility Results for Regular Languages in Nonregular Settings

Howard Straubing
Computer Science Department
Boston College
Chestnut Hill, Massachusetts
USA 02467

April 25, 2005

My ostensible purpose in this talk is to describe some new results (found in collaboration with Amitabha Roy) on expressibility of regular languages in certain generalizations of first-order logic. [10]. This provides me with a good excuse for describing some the work on the algebraic theory of regular languages in what one might call “nonregular settings”.

The syntactic monoid and syntactic morphism of a regular language provide a highly effective tool for proving that a given regular language is not expressible or recognizable in certain compuational models, as long as the model is guaranteed to produce only regular languages. This includes finite automata, of course. but also formulas of propositional temporal logic, and first-order logic, provided one is careful to restrict the expressive power of such logics. (For example, by only allowing the order relation in first-order formulas.)

Things become much harder, and quite a bit more interesting, when we drop this kind of restriction on the model. The questions that arise are important (particularly in computational complexity), and most of them are unsolved. They all point to a rich theory that extends the reach of algebraic methods beyond the domain of finite automata

1 Uniformizing Nonuniform Automata with Ramsey’s Theorem

Let’s start with an especially trivial application of the syntactic monoid: Let $\Sigma = \{0,1\}$, and consider the two languages

$$L_1 = \{w \in \Sigma^* : |w| \equiv 0 \pmod 2\}$$

and

$$L_2 = \{w \in \Sigma^* : |w|_1 \equiv 0 \pmod 2\}.$$

(We denote by $|w|_1$ the number of 1's that in the string w.)

These two languages have the same syntactic monoid, namely the group of two elements. It follows immediately that neither can be recognized by a finite automaton whose transition monoid is aperiodic (i.e., contains no nontrivial groups). Put another way, if $\phi : \Sigma^* \to M$ is a homomorphism onto a finite aperiodic monoid, then there is no $X \subseteq M$ such that $\phi^{-1}(X) = L_1$, or $\phi^{-1}(X) = L_2$.

Suppose that instead of a homomorphism, we consider a map

$$\psi : \Sigma \times \mathbf{Z}^+ \to M,$$

and extend it to $\Psi : \Sigma^* \to M$ by mapping

$$w = \sigma_1 \cdots \sigma_n$$

to

$$\Psi(w) = \psi(\sigma_1, 1)\psi(\sigma_2, 2) \cdots \psi(\sigma_n, n).$$

You can think of this as a kind of "nonuniform automaton", in which the state transition induced by a letter depends on the position of the letter within the input string. This obviously increases the computational power of the model; the languages recognized need not even be recursively enumerable! Can we recognize L_1 with such a setup? Can we recognize L_2?

The answer to the first question is "yes": Let M be the aperiodic monoid $\{1, a, b\}$ with multiplication defined by $Ma = a$, $Mb = b$. Set $\psi(\sigma, i) = b$ whenever i is odd, and $\psi(\sigma, i) = a$ when i is even. Then $L_1 = \Psi^{-1}(a)$.

But L_2 cannot be recognized. Let us suppose that we have some aperiodic monoid M and map Ψ that does recognize L_2. We will show that in spite of the unruliness of Ψ, we can tame it so that it behaves like a homomorphism on a large set of inputs: Since M is aperiodic, it satisfies some identity of the form $x^n = x^{n+1}$ for some n. Let $0 \leq i < j$, and let us color the segment (i, j) by the pair

$$(\psi(0, i)\psi(0, i + 1) \cdots \psi(0, j - 1), \psi(1, i)\psi(0, i + 1) \cdots \psi(0, j - 1)).$$

Ramsey's Theorem guarantees the existence of a sequence

$$i_1 < i_2 < \cdots < i_{2n+2}$$

such that each (i_j, i_{j+1}) is has the same color (m_0, m_1). Consider a string $w_1 \in \Sigma^*$ of length $i_{2n+2} - 1$ that has 1's in positions $i_1, \ldots, i_{n+1}$, and 0's elsewhere. Then

$$\Psi(w_1) = \Psi(0^{i_1 - 1})m_1^{n+1}m_0^n.$$

Let us change the last 1 in w_1 to 0, giving a new string w_2. We now have

$$\Psi(w_2) = \Psi(0^{i_1-1})m_1^n m_0^{n+1} = \Psi(0^{i_1-1})m_1^{n+1}m_0^n = \Psi(w_1),$$

but the numbers of 1's in the two strings differ by 1. So Ψ cannot recognize L_2.

We get the same conclusion if we allow M to contain nontrivial groups of odd order. We get a similar conclusion if we replace L_2 by the set of strings in which the number of 1's is divisible by $q > 1$: This language cannot be recognized by a map $\psi \times \mathbf{Z}^+ \to M$ if every group in M has cardinality relatively prime to q.

2 Programs over Finite Monoids

The results of the last section are due to Barrington and Straubing [5]. The "nonuniform automata" we considered are special cases of *programs over finite monoids.* These are defined as follows: With each integer $n > 0$ we associate a sequence of *instructions*

$$(\psi_{1,n}, i_1), \ldots, (\psi_{r_n,n}, i_{r_n}),$$

where each $i_j \in \{1, \ldots, n\}$ and each $\psi_{j,n}$ is a map from Σ into M. The value of the program Ψ on

$$w = \sigma_1 \cdots \sigma_n \in \Sigma^n$$

is

$$\Psi(w) = \prod_{j=1}^{r_n} \psi_{j,n}(\sigma_{i_j}) \in M$$

In other words, the program scans the input word in some haphazard order, possibly revisiting the same input letter many times. At each input letter, the program emits an element of M, which depends on both the letter itself and the instruction. The product of these elements determines whether w is accepted or not. We call the function $n \mapsto r_n$ the *length* of the program.

The nonuniform automata of the preceding section are programs that make a single scan over their input strings. In fact, [5] establishes a stronger result:

Theorem 2.1. *Let $q > 0$, and let $L \subset \{0,1\}^*$ be the set of strings in which the number of 1's is divisible by q. Let M be a finite monoid in which every group has cardinality relatively prime to q. Then any program over M recognizing L has length $\Omega(n \log\log n)$.*

If we allow the program length to be polynomial in the input length, however, we get the following remarkable result, due to Barrington [1]

Theorem 2.2. *If M is a finite monoid that contains a nonsolvable group,then every regular language is recognized by some polynomial-length program over M.*

This implies, for example, that Theorem 2.1 cannot be extended to polynomial-length programs, since the set of strings in which the number of 1's is divisible by 7 is recognized by the alternating group A_5, whose order is 60. In fact, Barrington showed that every language in the circuit complexity class NC^1 is recognized by a polynomial-length program over A_5. (Any other nonsolvable group will do.) It is this connection with circuit complexity that motivates the interest in programs over monoids.

3 Programs, Logic and Circuits

Let's take a look again at Theorem 2.2 and ask what happens when M contains only solvable groups. Such monoids are called *solvable monoids.* If the program into M is an ordinary homomorphism, then the language L recognized by the program is recognized by M in the ordinary sense; in particular, the syntactic monoid $M(L)$ of L is solvable. Straubing, Thérien and Thomas [13] give a characterization in generalized first-order logic of the regular languages recognized by finite solvable monoids: Consider the logic in which variables represent positions in a string over Σ, and which there are the following relation symbols: $x < y$, which is interpreted to mean that position x is to the left of positon y, and $Q_\sigma x$, where $\sigma \in \Sigma$, which is interpreted to mean that the letter in position x is σ. In addition to the usual boolean operations and ordinary quantification, we allow *modular* quantifiers $\exists^{r \bmod q}$, where $\exists^{r \bmod q} x\phi$ is interpreted to mean "the number of positions x satisfying ϕ is congruent to r modulo q.

A sentence in this logic accordingly defines a language over Σ. For instance, the formula $\phi(x)$ given by

$$\exists y(x < y \wedge \neg\exists z(x < z \wedge z < y) \wedge Q_\sigma x \wedge Q\sigma y)$$

says "position x and its successor both contain the letter σ", and thus the sentence

$$\exists^{0 \bmod 2} x\phi(x)$$

defines the set of strings in which $\sigma\sigma$ occurs an even number of times as a factor. In [13] it is proved that the family of languages defined in this way is precisely the family of regular languages over Σ recognized by finite monoids in which every group is solvable.

Suppose now that instead of a homomorphism, we have a polynomial-length program over M. On an input of length n, the program emits a sequence

$$m_1, m_2, \ldots, m_{n^k}$$

of elements of M. We can view this sequence as an element w' of M^*, where we treat M as a finite alphabet. We then have $w \in L$ if and only if $w' \in L'$, where L' is the set of strings in M^* that multiply to a value in X.

Of course, L' is a regular language recognized by M, so by the theorem just cited, L' is defined by a sentence ϕ' using both modular and ordinary quantifiers, and in which the only numerical predicate is $<$.

In [2] it is shown how we can rewrite this sentence to obtain a defining sentence ϕ for L The essential idea is that each position in the w' can be encoded by a k-tuple of positions in w, and thus each variable in ϕ' is replaced by a k-tuple of variables in ϕ. However, in constructing the sentence for L, we are obliged to introduce new numerical predicates that encode the nonuniform behavior of the k-program Ψ. This is no surprise: since k-programs over M can recognize uncomputable languages, so we would necessarily introduce uncomputable numerical predicates in the defining sentences.

Thus every language recognized by a polynomial-length program over a solvable monoid is definable by sentence with modular quantifiers (with no restriction on the kinds of numerical predicates introduced into the sentence).

Conversely, every language defined by such a sentence is recognized by a polynomial-length program over a finite solvable monoid. Thus we have:

Theorem 3.1. *$L \subseteq \Sigma^*$ is defined by a sentence using modular and ordinary quantifiers if and only if L is recognized by a polynomial-length program over a finite solvable monoid.*

Moreover, these are exactly the languages recognized by a special kind of boolean circuit: polynomial-size, constant-depth families of circuits with unbounded fan-in gates to compute AND, OR, and MOD_q, for a modulus $q > 1$ fixed throughout the family. In computational complexity theory, this class of languages is called ACC. The connection between ACC and programs over solvable monoids was discovered by Barrington and Thérien [6]. We will not discuss circuits further here, but instead concentrate on the representations of ACC in terms of programs and logic.

4 The Main Question

As we've seen, when we do not restrict the numerical predicates that occur in our quantified formulas, we obtain languages that have arbitrarily high computational complexity, in the sense that they may be uncomputable, but have tightly bounded computational complexity in quite a different sense, since they are recognized by small circuits. However, the true computational power of such circuits is an open question, since we do not even know how to show that ACC does not contain NP-complete languages.

But what if a language L defined by such a sentence is known to be regular? Let us first consider the case of a regular language defined by a first-order sentence, without the use of modular quantifiers. The example given in the first section shows that the syntactic monoid of such a language might contain a group. Indeed, the sentence

$$\forall x(\forall y(y \leq x) \rightarrow (x \bmod 2 = 0))$$

defines the set of strings of even length. Let us suppose, however, that the input alphabet Σ contains a *neutral letter* for L—that is, we suppose there exists $\sigma \in \Sigma$ such that σ is mapped to the identity element of the syntactic monoid of L. This rules out such examples as the set of strings of even length. We can then show

Theorem 4.1. *Let $L \subseteq \Sigma^*$ be a regular language such that Σ contains a neutral letter for L. If L is defined by a first-order sentence, then $M(L)$ is aperiodic.*

This theorem is due to Barrington, *et. al.* [2]. The proof, however, requires the solution of a difficult problem in circuit complexity. (The separation of AC^0 from NC^1, see Furst, Saxe and Sipser [7].)

We do not know how to show that there is any regular language not definable by a sentence with modular quantifiers. But it has long been conjectures that nonsolvability is necessary for the behavior observed in Theorem 2.2.

Conjecture 4.2. *Let $L \subseteq \Sigma^*$ be a regular language defined by a sentence that inlcudes both modular and ordinary quantifiers. Suppose that Σ contains a neutral letter for L. Then every group in $M(L)$ is solvable.*

This conjecture is equivalent to the assertion that the circuit complexity class ACC is properly contained in the class NC^1, a long-unsolved problem in computational complexity. (The neutral letter hypothesis is not, strictly speaking, necessary, since if the conjecture is true in the above form it remains true without the hypothesis.)

There is, as well, a purely modular form of the conjecture:

Conjecture 4.3. *Let $L \subseteq \Sigma^*$ be a regular language defined by a sentence that inlcudes only modular quantifiers. Suppose that Σ contains a neutral letter for L. Then $M(L)$ is a solvable group..*

How might we approach such a question? The arguments in Section 1 show that for languages defined by single-scan programs, we can use Ramsey's Theorem to smooth out the non-uniformity in the program and make it look like a homomorphism. Can we apply the same ideas to logical formulas? Perhaps we can prove the conjectures for special classes of formulas.

If the only numerical predicate occurring in our formulas is the order relation, then Conjectures 4.2 and 4.3 hold because of the results, already cited, in [13]. But this is what we have been calling a *regular setting:* Formulas such as these cannot define nonregular languages. The simplest example of a nonregular setting is provided by formulas in which, in addition to the ordering relation, there are *monadic* numerical predicates (predicates with a single argument). The following is from Straubing [11]:

Theorem 4.4. *Conjectures 4.2 and 4.3 hold for formulas in which every numerical predicate is the order relation or a monadic relation.*

This holds because such formulas give rise to single-scan programs with equivalent behavior. More general formulas give rise to programs whose length is n^k for $k > 1$, and uniformizing these is considerably more difficult.

5 Presburger Arithmetic and Active-Domain Sentences

Presburger Arithmetic is the first-order theory of the natural numbers with addition. In other words, a formula in this theory is just a first-order formula (typically with free variables) with the single ternary predicate $x = y + z$. A formula such as

$$\exists y(x = y + y)$$

expresses the property "x is even". Of course, we need quantification to express such a property. Suppose though, that we add some new symbols to our logic: Specifically, we allow constants 0 and 1, and atomic formulas $t \equiv_m t'$, where t and t' are terms, interpreted to mean t and t' are congruent modulo m. We can express "x is even" by the quantifier free formula

$$x \equiv_2 0.$$

Presburger [9] showed that every formula in the original logic is equivalent to a quantifier-free formula, provided we add the constants 0 and 1, extend the set of relations to include ordering and all the formuals $t \equiv_m t'$. (Since these can all be expressed in the original logic, we do not change the properties definable in Presburger arithmetic by adjoining them.)

We can use the apparatus of Presburger arithmetic to define languages: Let us add a single unary predicate symbol π. We interpret $\pi(x)$ to mean "the bit in position x is 1". (In other words, $\pi(x)$ has the same meaning as Q_1x, but it is important in this context that there is no Q_0x.) A sentence in this logic can be interpreted in a string of bits and accordingly defines a language in $\{0,1\}^*$. We also allow interpretation in infinite strings in $\{0,1\}^*0^\omega$, that is, infinite bit strings in which there are finitely many 1's.

We say that a sentence in this logic is an *active-domain* sentence if every quantifier occurs in the form

$$\exists x(\pi(x) \wedge \psi).$$

In other words, we only allow quantification over positions that contain a 1.

The same techniques used to prove quantifier elimination in Presburger arithmetic can be used to show that every sentence in this logic is equivalent to an active-domain sentence, provided we extend the signature to include 0,1, ordering, and congruence modulo k. A proof is outlined in Libkin [8].

When we have modular quantifiers available as well, active-domain quantification means that every modular quantifer occurs in the form

$$\exists^{r \bmod q} x(\pi(x) \wedge \psi).$$

Very recently, A. Roy and I [10] showed that the elimination of non-acitive-domain quantifiers can be extended to formulas that contain modular as well as ordinary quantifers:

Theorem 5.1. *Let $L \subseteq \{0,1\}^*$ or $L \subseteq \{0,1\}^*0^\omega$. Suppose L is defined by a sentence ϕ, with both modular and ordinary quantifiers and with relation symbols $+$ and π. Then L is defined by an active-domain sentence ϕ' with modular and ordinary quantifers, constant symbols 0 and 1, and relation symbols $+$, $<$ and $\equiv_m$. Moreover, the moduli of the modular quantifiers in ϕ' all occur in ϕ.*

6 Ramsey's Theorem Again

We can talk about active-domain sentences with reference to arbitrary alphabets Σ of input letters, not just the binary alphabet $\{0,1\}$. We simply designate some $\tau \in \Sigma$ to be the "inactive letter", and require all quantifiers to occur in the context

$$\mathcal{Q}x(\bigvee_{\sigma\neq\tau} Q_\sigma x \wedge \phi),$$

where $\mathcal{Q}$ is either an ordinary existential quantifier, or a modular quantifier.

Theorem 6.1. *Conjectures 4.2 and 4.3 are true for languages defined by active-domain sentences in which the neutral letter is inactive. Moreover, every such language is itself regular and its syntactic monoid is a solvable monoid (in Conjecture 4.2) or a solvable group (in Conjecture 4.3).*

There is no restriction in the foregoing theorem on the numerical predicates that can occur in the sentence. Combining this with Theorem 5.1, we obtain the main result of [10]:

Theorem 6.2. *Conjecture 4.2 is true for languages defined by sentences in which the only numerical predicate symbol is* $+$.

Let us sketch how Theorem 6.1 is proved. Once again, we can use Ramsey's theorem to uniformize a non-uniform computation. This time the uniformization functions at the level of the formula, instead of the program. Following Libkin [8], we were able to prove:

Theorem 6.3. *Let* $L \subseteq \Sigma^*$. *Suppose* $L0^\omega$ *is defined by an active-domain sentence* ϕ *with both modular and ordinary quantifiers, and arbitrary numerical predicates. Then there is an infinite subset* Y *of* $\mathbf{N}$, *and a sentence* ψ, *with the following properties: (a) The only numerical predicate in* ψ *is* $<$. *(b)* ψ *has both ordinary and modular quantifiers, and the modulus of every modular quantifier in* ψ *also occurs in* ϕ. *(c)* ψ *is an active-domain sentence. (d) If* $w \in \Sigma^*$ *and all the active letters in* w *are in positions belonging to* Y, *then* $w \in L$ *if and only if* w *satisfies* ψ.

In other words, nonuniform sentences (those with unrestricted numerical predicates) behave exactly like highly uniform sentences (those in which the only numerical predicate is $<$) when restricted to some large set of positions. This is precisely analogous to the behavior we observed in Section 1.

Now let $L \subseteq \Sigma^*$ be a language with a neutral letter defined by an active-domain sentence ϕ in which the neutral letter τ is inactive. Then $L0^\omega$ is also defined by ϕ, and thus there is an infinite subset Y of $\mathbf{N}$ and a sentence ψ as in the theorem above. Let

$$Y = \{y_1 < y_2 < \cdots\}.$$

Let $w = \sigma_1 \cdots \sigma_n \in \Sigma^*$, and consider the word w' of length y_n such that the y_i^{th} letter of w' is σ_i, and all the other letters are equal to the neutral letter τ. Then $w \in L$ if and only if $w' \in L$ if and only if w' satisfies ψ. But since ψ uses only active-domain quantification and has only $<$ for a numerical predicate, this occurs if and only if w satisfies ψ. This means that L is regular and its syntactic monoid contains only solvable groups.

7 Conclusions

Much of what we have described can be viewed as a generalization of the work in Barrington, *et. al.* [4] on the "Crane Beach Conjecture" from the domain of first-order formulas with $+$ to formulas with modular quantifiers and $+$. The outstanding challenge, of course, is to extend our non-expressibility results to formulas with arbitrary numerical predicates.

There are, however, considerable technical obstacles to doing so. We might try to approach the problem by first showing that Conjecture 4.2 holds when we adjoin the numerical predicate $\times$ (integer multiplication). (The resulting logic is important from a computational standpoint, since it defines precisely the languages in a natural uniform version of ACC [3].) But in [4] it is shown that when this is done, one can define languages with neutral letters that are not regular. Together with Theorem 6.1 this implies, in particular, that one cannot eliminate non-active-domain quantification from such a formula. Quantifier-elimination techniques may still be useful, however—it would be enough to show the reduction to active-domain quantification under the assumption that the language defined is regular. It may be necessary to begin with very simple formulas: In Straubing [12] something of the kind is carried out for the modular analogue of boolean combinations of Σ_1 sentences. Once again, Ramsey theory is an essential ingredient.

Results like Theorem 2.2 show that nonsolvable groups have special computational properties. What we have been trying to do is show that the nonsolvability is essential—that monoids with only solvable groups have radically different computation capabilities. Completing this program would extend the application of finite semigroups in computation well beyond the domain of finite automata.

8 References

References

[1] D. Mix Barrington, "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1", *J. Comp. Syst. Sci.* **38** (1989) 150–164.

[2] D. Mix Barrington, K. Compton, H. Straubing, and D. Thérien, "Regular Languages in NC^1", *J. Comp. Syst. Sci.* **44** (1992) 478–499.

[3] D. Mix Barrington, N. Immerman, and H. Straubing, "On Uniformity in NC^1", *J. Comp. Syst. Sci.* **41** (1990) 274–306.

[4] D.M. Barrington, N. Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien The Crane Beach Conjecture, LICS '01, 187-196.

[5] D. Mix Barrington and H. Straubing, "Superlinear Lower Bounds for Bounded-Width Branching Programs", *J. Comp. Syst. Sci.* **50** (1995) 374–381.

[6] D. Mix Barrington and D. Thérien, "Finite Monoids and the Fine Structure of NC^1", *JACM* **35** (1988) 941–952 .

[7] M. Furst, J. Saxe, and M. Sipser, “Parity, Circuits and the Polynomial-time Hierarchy”, *Math. Systems Theory,* **17** (1984) 13-27.

[8] L. Libkin, “Embedded finite models and constraint databases”, in E. Grädel, et. al., eds., *Finite Model Theory and its Applications,* Springer, New York (2005).

[9] M. Presburger, “Ueber die Vollstaendigkeit eines gewissen Systems der Arithmetik ganzer Zahlen”, in welchem die Addition als einzige Operation hervortritt.” In Comptes Rendus du I congrs de Mathmaticiens des Pays Slaves. Warsaw, Poland: pp. 92-101, 1929.

[10] A. Roy and H. Straubing, ‘Definability of Languages by Generalized First-ordr Formulas over (**N**,+)”. Preprint.

[11] “Circuit complexity and the expressive power of generalized first-order formulas”. In Proceedings 1992 ICALP *LNCS* **623** (1992) 16–27.

[12] H. Straubing, “Languages defined with modular counting quantifiers”. Inform. and Comput., **166** (2001) 112-132..

[13] H. Straubing, D. Thérien, and W. Thomas, “Regular Languages Defined with Generalized Quantifiers”, *Information and Computation* **118** (1995) 289-301.