ON PARIKH MAPS, ABSTRACT LANGUAGES AND DECISION PROBLEMS

Alfred Borm and Louis E. Rosier

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-84-22 July 1984

Table of Contents

Abstract

We consider formulas similar to Presburger formulas, but extended by allowing the predicate "|", (for divides); but not allowing universal quantification of the resulting expressions. The solution sets for such formulas are known to be not semilinear, in general. We show that some results concerning decidability questions for classes of languages with effectively constructible semilinear Parikh maps can be extended to languages whose effectively constructible Parikh maps are solutions to these new formulas, which we call "modular". We suggest that modularity may offer a natural extension of semilinearity, in which many previously-established results may remain valid.

1. Introduction and Definitions

The study of abstract languages has been an active area of research during recent decades, as the community of theoretical computer scientists strives to explore the inherent limitations on what machines are capable of doing. The landmark works of Turing [22], Chomsky [1], and Rabin and Scott [17] served as major foundations for the many results that followed.

An important technique is due to Parikh [16]. This technique uses *Parikh maps* (technical definitions will come later in the paper) which record the frequency of occurrence of each character in the alphabet in use, in each word. For certain classes of languages, including all context-free languages [16][1] , the images of the languages in the class under the Parikh map all have an algebraic property called *semilinearity*. This makes their analysis possible in terms of *Presburger formulas* and their related solution sets. Exploiting this link between formal languages, algebra and mathematical logic has been a major tool for achieving results regarding the decidability of questions about abstract languages, when these languages can be shown to have semilinear Parikh maps.

For example, let $\mathcal{L}$ be a family of languages which is effectively closed under inverse homomorphism and intersection with a regular set. If, in addition, the languages in $\mathcal{L}$ have effectively constructible semilinear Parikh maps then it can be shown [8] that there exists an algorithm to decide given an arbitrary language in $\mathcal{L}$ and a language accepted by a one-way nondeterministic reversal-bounded multicounter machine [10] whether their intersection is empty. In [8], this result was used to show that there is an algorithm to decide given a language $L \in \mathcal{L}$ and two 2-way deterministic sequential transducers (2DST's) whether they are equivalent on L. (Furthermore, it was shown, in [8] that, in general, the semilinearity requirement could not be removed.)

It is the purpose of this paper to propose a larger class of formulas and related sets, to be called *modular formulas* (and sets). This class will be shown to properly include the Presburger formulas, and sets. The use of this class will allow several previously-obtained results about decidability to be extended to a wider class of languages. Some examples of such extensions are provided in this paper.

More specifically, in this paper we examine the class of languages accepted by 2-way deterministic counter machines with a single bounded-reversal counter. Bounded such languages were shown to have a decidable emptiness problem in [4]. Later the result was generalized to the unbounded case in [6]. Using essentially the argument provided in [6], we show that these languages have effectively constructible modular Parikh maps. Since this class of languages is also closed under inverse homomorphism and intersection with a regular set, it is natural to ask whether the results of [8], mentioned above, extend in this case. One might expect not, since the Parikh maps are not, in general, semilinear. However, we show that the results do generalize. In fact, we show that the results of [8], mentioned in the previous paragraph, generalize to any family of languages which is effectively closed under inverse homomorphism and intersection with a regular set, when the languages have effectively constructible modular Parikh maps. The techniques in [8] can be modified to show the extensions. The following definitions and known results will be used throu-

---

[1] Other examples of such language classes can be found in [13, 11, 18, 20, 10, 5, 3, 21, 12].

ghout the remainder of the paper. Other terms of formal language theory may be found in Harrison [7].

Definition: An *elementary expression* is any expression of the form:

$$t_0 + \sum_{i=1}^{n}(t_i x_i)$$

where each t_j, j=0,...,n is an integer constant, and each x_k, k=1,...,n, is a variable over N, the set of non-negative integers.

Definition: The class of *elementary Presburger formulas* is the smallest class which contains all formulas of this type:

$$E_1 \quad P \quad E_2$$

and which is closed under negation, conjunction, and disjunction. Here E_1 and E_2 are elementary expressions, and P is any of these predicates: $=, \neq, <, \leq, >, \geq$.

Definition: A *Presburger formula* is any formula of this type:

$$Q(F)$$

where F is an elementary Presburger formula, and Q is a (possibly empty) sequence of existential quantifiers.

Notation: In the text to follow, for any formula (Presburger or not) Q(F) in which no quantifiers appear in F, and which includes free variables $x_1,...,x_n$ and bound variables $y_1,...,y_m$ (the variables on the quantifiers in Q), we denote that briefly by writing:

$$F(x_1,\ldots,x_n \ ; \ y_1,\ldots,y_m).$$

The preceding definition for Presburger formulas is not the standard one, since it does not allow universal quantifiers. However, each Presburger formula $F_1(x_1,..,x_n \ ; \ y_1,..,y_m)$, in the sense of the usual definition, is equivalent to a Presburger formula in the sense of this definition, $F_2(x_1,..,x_n \ ; \ y_1,..,y_k)$, for some $k \leq m$. These two formulas will have the same free variables. Since we are primarily interested in the subsets of N^n satisfying such formulas, the given definition is sufficient.

The Presburger formulas are known to be closely related to certain sets of vectors in N^N, defined below.

Definition: A subset, A, of N^n is called *linear* iff there exist $v_0, v_1, \ldots, v_r$ in N^n such that:

$$A = \{ v_0+k_1v_1+ \ldots +k_rv_r \mid k_i \geq 0,\ i=1,\ldots r\}$$

A *semilinear set* is any finite union of linear sets.

Definition: If $F(x_1,\ldots,x_n\ ;\ y_1,\ldots y_m)$ is a formula as described above, the *solution set* of F is:

$$\{ \langle x_1,\ldots,x_n\rangle \mid F(x_1,\ldots,x_n\ ;\ y_1,\ldots,y_m\ ) \text{ is true}\}.$$

We denote this set S(F).

The following major results relate to the uses of these concepts.

Theorem: A subset of N^n is semilinear iff it is the solution set of some Presburger formula [2].

Definition: Let $\Sigma = \langle a_1,\ldots,a_n\rangle$ be an alphabet whose symbols are arranged in an arbitrary, fixed order; and let L be a language over Σ. For any w in L, the *Parikh map* of w is defined as:

$$f_\Sigma(w) = \langle k_1,\ldots,k_n\rangle$$

where, for each i, i=1,...,n, a_i occurs k_i times in w. The Parikh map of L is defined by:

$$f_\Sigma(L) = \{ f_\Sigma(w) \mid w \in L \}.$$

The famous result known as *Parikh's Theorem* [16] states that the Parikh map of every context-free language is a semilinear set. It is easy to see that the converse is false, however, since $\{a^nb^nc^n \mid n\geq 1\}$ is a language that is not context-free, but whose Parikh map is the solution set for the Presburger formula $(x_1=x_2)\wedge(x_2=x_3)\wedge(x_1\geq 1)$.

The *emptiness problem* for a language is the problem of determining if the language is empty. Similarly, the *finiteness problem* pertains to determining if a given language is finite or not. It is known [2] that each of these problems is decidable for languages with effectively constructible semilinear Parikh maps.

Our extension to the preceding is as follows.

Definition: The class of *elementary modular formulas* is the smallest class containing all formulas of this type:

$$E_1 \quad P \quad E_2$$

and which is closed under negation, conjunction, and disjunction. Here E_1 and E_2 are any elementary expressions; and P is any of these predicates: $=, \neq, <, \leq, >, \geq, |$; where "|" stands for "divides."

Definition: A *modular formula* is any formula of this type:

$$Q(F)$$

where F is an elementary modular formula and Q is a (possibly empty) sequence of existential quantifiers.

Definition: A *modular set* is any subset of N^n, for some n, which is the solution set for some modular formula.

It is known [4, 14] that such sets are not, in general, semilinear.

Definition: A language, L, over an alphabet, Σ, is a *modular language* iff $f_\Sigma(L)$ is a modular set.

Definition: A language, L, over an alphabet $\Sigma=<a_1,...,a_n>$, in some arbitrary, fixed order, is called *bounded* iff $L \subseteq a_1^* a_2^* ... a_n^*$.

Definition: If M is any finite automaton, T(M) denotes the language accepted by M.

Definition: A *marked regular set* is a regular set in which each string has the same distinguished initial character.

2. A Class of Modular Languages

Gurari and Ibarra [4] examined the class of bounded languages accepted by 2-way deterministic counter machines with one bounded-reversal counter. These are deterministic machines having an end-marked input tape which can be read repeatedly in either direction. The finite control of the machine is augmented by a single stack which is restricted to only two stack symbols, a bottom marker, 0, and another symbol, 1. This stack, or counter, is further restricted in that there is a predetermined bound on the number of times the stack contents, viewed as an integer, can be changed from increasing to decreasing, or *vice versa*. Such a machine will, for brevity, be called a CM.

The major result in [4] is that the emptiness question for that class of languages is decidable. The result is obtained by demonstrating a system of Diophantine equations which has a solution if and only if some word is accepted by the machine in question; and then applying a result of Lipschitz [14] that it is decidable whether a particular formula of the type derived has a solution. Thus these machines furnish an example of a class of languages having a decidable emptiness problem, but whose Parikh maps are not, in general, semilinear (as is noted in [4]).

Gurari [6] extended this result to the case of unrestricted (i.e. not bounded) languages recognized by these machines. The result is unpublished, and is presented here in order to clarify the rather sketchy proof in the available draft copy, and to alter it to fit the language of this paper. We have varied some of the details of the construction, in the interest of clarity of presentation.

The CM's are assumed here to begin at the left end of the input string, with 0 in the counter, and to halt after entering any accepting state. On each move, a CM can change the state of its finite control, move the input head one position either way (or not), and change the counter contents by +1, -1, or 0. It is understood that the counter cannot be decreased when it contains 0, and the input head cannot be moved past either end marker on the input tape.

At any time the counter is said to be in: *zero status* if the current transition began with the counter at 0; *increasing status* if the counter is positive and the last transition performed which changed the counter increased it; and *decreasing status* if the last transition that changed the counter decreased it. During the course of a computation, the machine begins in mode 1, and the mode increases by 1 each time the status changes.

Gurari proposes a nondeterministic algorithm to construct all possible time-space graphs representing canonical accepting computations for such a machine. Each node in the graph is an ordered triplet $<\alpha$,n,C$>$, where α is a transition rule of the machine, n is an integer representing the mode value during the interval from the occurrence of α until just before the next transition, and C is a character in {'Z','I','D'}, which indicates if the machine is in zero, increasing, or decreasing status during that interval. The nodes are arranged in columns, each column headed by the input character expected by the transitions in the nodes in that column.

A vertical column in such a time-space graph is called a *cut*. That is, a cut is the sequence of nodes appearing at a given input character, for the computation represented by the graph. Since the number of counter reversals is bounded, so is the number of mode changes, hence also is the number of modes. The number of transitions is finite, of course. So, if there is some way to limit the number of repetitions of a given node in a cut, the number of possible cuts for a given machine will be finite.

Gurari achieves this limitation on repetition by a technique he calls *folding*. To understand this technique, we must examine the structure of the graph in detail.

We require the following labeling rules for the nodes in a completed graph:

1. **If** the transition must have 0 in the counter to be performed, any node with that transition must be labeled 'Z'; **elseif** the transition increases the counter, the node must be labeled 'I'; **elseif** the transition decreases the counter, the node must be labeled 'D'; **else** the node must have the same label as its predecessor.
2. The initial node is labeled 'Z', mode=1.
3. Any other node which is labeled the same as its predecessor, must have the same mode value also.
4. Any other node which is not labeled the same as its predecessor, must have a mode 1 larger than that of its predecessor.

Now, suppose a cut contains two nodes that are identical. This would indicate a computation caught in a loop, returning to the same input character in the same control state and in the same mode. If the node is labeled "I" or "Z," the construction should be abandoned, because the only way for such a loop to terminate is for the counter to decrease to zero. This can only hap-

pen with a node labeled "D." We have arranged things so all nodes in a given mode are labeled the same. Thus we see several important points: a single mode can only contain one such loop; that loop can terminate only at the end of the mode; the counter moves in a monotone manner in each mode; and the first mode cannot contain a loop, since only decreasing modes can be allowed to have loops.

Gurari's technique of folding consists of observing if the last node in a given mode is repeated in the cut in which it appears. If it is, he retains only one cycle of the loop in the mode, and identifies the first and last occurrences of that last node. As a result, no single node can occur in a given cut more than three times (it may occur once in the cut before the loop begins, and twice in the loop). This ensures that the number of possible cuts for a given machine is finite.

The algorithm thus proceeds to construct, nondeterministically, sequences of possible cuts, from left to right. If any branch of the algorithm finds that a connected graph has been constructed that leads from the start state to an accepting state and obeys the consistency rules above, it halts and reports its results. If a branch discovers that its incomplete graph cannot be extended, but does not meet those conditions, it halts and rejects.

The termination of this construction is guaranteed by the following technique. As the algorithm discovers each new cut to add to the sequence, X, of cuts being formed, it checks to see if the new cut is already present in the sequence X. If it is not, it is marked as new. If it is already present, and no cut between it and its previous occurrence is marked, then the whole sequence subsequent to the preceding occurrence of that cut is removed, and placed in another set, called S. This second set, S, contains all these repeated sequences of cuts. The sequences in S represent input strings which cause the machine to cycle through some states, and emerge with exactly the same pattern of states and transitions in the graph as when the subsequence was begun.

It follows that each branch of this algorithm must halt, either by abandoning its construction, or by finding an accepting graph; because each is limited to a finite number of choices. The result of this process is a finite number of (X,S) pairs. Each one represents a canonical form for a string that may be accepted by the machine.

If $X=\alpha_1\alpha_2...\alpha_r$, then each sequence, $B_i = \beta_1\beta_2...\beta_t$ in S can be inserted into X at any point immediately to the right of any occurrence of β_t in X, and form a consistent graph. These S-sequences may need to be repeated an unknown number of times to enable acceptance by the machine. In order to determine if such a repetition pattern is possible, a system of modular formulas can be developed, whose solution guarantees that some version of the (X,S) pattern will be accepted.

We note first that, since the algorithm is exhaustive, if there is a string accepted by the machine, its acceptance path must occur as one of the (X,S) pairs found by the algorithm. Thus, if there are no (X,S) pairs found, the emptiness problem is solved for the given machine, since nothing can be accepted.

Suppose, then, that we have an (X,S) pair selected by the algorithm. If the alphabet, $\Sigma = <u_1,u_2,...,u_n>$ in some fixed order, and X contains g different modes, and S contains f different

sequences in some fixed order. Let $w_1,w_2,...,w_n$ represent the number of times the characters in Σ appear in the accepted string. Let $x_1,x_2,...,x_f$ be the number of times the separate sequences of S are to be repeated in the string; and let $y_1,y_2,...,y_g$ represent the values in the counter when the last transition in each mode has been performed.

Then the behavior of the counter in an accepting computation must satisfy these formulas:

(1) $y_i = 1$

For any subsequent mode i, i>1, which does not contain a loop:

(2) $y_i = y_{i-1} + b_i + \sum_{j=1}^{f} a_{ij}x_j$.

If i>1 and the nodes in mode i are labeled 'Z', then:

(3) $y_{i-1} = 0$

Here b_i is the change in the counter due to the transitions in X which are in mode i, and a_{ij} is the change in the counter due to the transitions in mode i that are in the jth sequence of S.

For any mode i, i>1, that does contain a loop, we add four formulas:

(4) $y_i = 0$

since the only way to exit a loop is to drive the counter to zero; and:

(5) $y_{i-1} + \underline{b}_i + \sum_{j=1}^{f} \underline{a}_{ij}x_j > 0$

and

(6) $(d_i + \sum_{j=1}^{f} c_{ij}x_j) \mid (y_{i-1} + \underline{b}_i + \sum_{j=1}^{f} \underline{a}_{ij}x_j)$

since the counter must be positive before it can be decreased to zero, and the loop must end exactly when the counter reaches zero. Also:

(7) $d_i + \sum_{j=1}^{f} c_{ij}x_j < 0$

since each iteration of the loop must reduce the counter.

In this case, $\underline{b}_i$ is the change in the counter due to the transitions in mode i that are in X and not in the loop; $\underline{a}_{ij}$ is the change in the counter due to the transitions in mode i that are in the jth sequence in S and not in the loop; and d_i and c_{ij} are the corresponding two figures for the transitions in mode i that are in the loop.

To relate all this to the Parikh map of the language, we add, for k=1,...,n:

$$(8)\quad w_k = r_k + \sum_{j=1}^{f} t_{kj}x_j$$

where r_k is the number of times the character u_k occurs in the cuts in X, and t_{kj} is the number of times it appears in the cuts in the jth sequence in S.

It is fairly clear that, if a string is accepted by the machine, it must be a representation of some (X,S) pattern, and the formulas written above must have a solution which can be found by tracing the acceptance path of the machine for that string.

Conversely, suppose for some (X,S) pair, the system of formulas has a solution. We build a string by inserting the sequences from S into X the number of times represented by the solution. Since the machine is deterministic, it has to follow the path shown by the cuts from (X,S), corresponding to this recreated string. It will start with its counter at zero, and, at the end of the first mode, must have the counter value shown in formula (1).

Subsequently, if the counter has achieved the proper value, y_{i-1}, at the end of mode i-1, then it must achieve the value y_i at the end of mode i. This is obvious from formula (2) if the mode has no loop, since one only accumulates the monotone counter changes that occur during the time of that particular mode. If the mode is labeled 'Z', formula (3) ensures that it can be performed. (For the first mode, of course, this is guaranteed by the starting conditions of the CM.)

If there is a loop, then that must be the last thing in the mode. The counter will achieve the positive value called for in formula (5), and formula (6) ensures that this value is a multiple of the decrease in the loop provided by formula (7). So the counter must reach exactly zero at the end of some iteration of the loop, as predicted by formula (4).

Hence, the machine will be able to exactly retrace the computation in each mode, arriving at an accepting state. Thus the machine accepts some string if and only if the algorithm produces an (X,S) pair, with its associated set of formulas, and there is a solution for the set of formulas. To decide the emptiness question for a particular, CM, then, one uses Gurari's algorithm to produce a finite number of (X,S) pairs, each with its associated formulas. Take the conjunction of the formulas for each particular (X,S) pair, and the disjunction of these over all (X,S) pairs to produce a single formula which has a solution if and only if some word is accepted by the machine. Since Lipschitz [14] has shown that testing this formula for a solution is a decidable problem, the emptiness problem for this class of machines is decidable.

Thus we have:

Theorem 2.1: The languages accepted by CM's have effectively constructible modular Parikh maps.

Proof: The formula produced by the algorithm is modular. If we represent that formula by:

$$F(w_1,\ldots,w_n,x_1,\ldots,x_f,y_1,\ldots,y_g)$$

then the Parikh map of the language in question must be the solution set for:

$$F(w_1,\ldots,w_n\ ;\ x_1,\ldots,x_f,y_1,\ldots,y_g)$$

which is obtained by providing existential quantifiers for the variables in F which are not part of the Parikh map.□

3. Some Closure Properties of Modular Sets

The following closure properties of modular sets are of general interest, and are useful in the applications in the following chapter. The following result can be easily verified.

Theorem 3.1: The projection of a modular set is an effectively-constructible modular set.

Definition: We let $\underline{Md}_n$ denote the class of all modular sets of the form S(F), where F is a modular formula with exactly n free variables.

The second part of the following result shows an important difference between modular sets and semilinear sets.

Theorem 3.2:

1. For a given $n \in N$, $n \geq 1$, Md_n is effectively closed under union and intersection.
2. In general, Md_n is not closed under complementation.

Proof: The first part of the theorem can be easily verified. Essentially, complementation allows us to represent formulas with both universal and existential quantifiers. This, in turn, allows the representation of all recursively enumerable sets [19]. For a simple proof, however, consider the following.

We know by [14] that it is a decidable question if a given modular formula has any solutions. If we consider the following formula:

$$F(x,y,z\ ;\ w)=\exists w(\neg(x|z)\vee\ \neg(y|z)\vee\ ((x|w)\ \wedge\ (y|w)\ \wedge\ \neg(z|w)\ )\ )$$

and let C=S(F); then the complement, C^c, can be found by:

$$C^c = S(\neg F)$$

Now,

$$\neg F(x,y,z\ ;\ w) = \forall w((x|z) \wedge (y|z)\ \wedge\ (\ \neg(x|w)\ \vee\ \neg(y|w)\ \vee\ (z|w))).$$

So we observe that:

$$S(\neg F) = \{\ \langle x,y,z\rangle\ |\ \ z = l.c.m.(x,y)\ \}.$$

If this is a modular set, then so is the following:

$$\{\langle x_1,x_2,x_3,x_4\rangle | ((x_3=x_1+1)\wedge(x_4=l.c.m.(x_1,x_3))\wedge(x_2+x_1=x_4))\ \text{is true}\}.$$

If, by projection, we eliminate the last two dimensions from this set, we have that:

$$\{\langle x_1,x_2\rangle\ |\ x_2 = (x_1)^2\ \text{is true}\ \}$$

is a modular set.

Given that, it follows that:

$$\{\langle x_1,\ldots,x_7\rangle|$$

$$(x_4=(x_1)^2 \wedge x_5=(x_2)^2\ \wedge\ x_6=x_1+x_2\ \wedge\ x_7=(x_6)^2\ \wedge\ x_4+x_5+2x_3=x_7)\ \text{is true}\}$$

is also modular. Analysis of the algebra in the formula above shows that the projection to the first three dimensions of this set gives:

$$\{\langle x_1,x_2,x_3\rangle\ |\ x_3 = x_1*x_2\ \text{is true}\ \}$$

is modular. But this can't be, since the ability to simulate multiplication implies the ability to make a modular set of:

$$\{\langle x_1,x_2,\ldots,x_k\rangle|\ P(x_1,\ldots,x_k) = 0\ \text{is true}\ \}$$

where P is an arbitrary polynomial. Thus the decidability of the emptiness problem for modular sets would imply the decidability of Hilbert's tenth problem, which would be a contradiction [15]. It follows that Md_n cannot be closed under complementation.□

4. An Application Example

Ibarra [8] discusses several decidability questions for some general classes of languages which have effectively constructible semilinear Parikh maps. We retrace those results here in order to illustrate how modularity can sometimes be used to extend results obtained using semilinearity arguments.

In the following theorems, whenever Ibarra specifies that a set should be semilinear, we have substituted the word "modular." The first theorem was shown in [10] for context-free languages only. Part of the theorem appearing in [8] is omitted here. We are unable to show decidability of the finiteness question for modular languages, hence we restrict ourselves to the question of emptiness.

Theorem 4.1 (Theorem 1)[2]: Let $\mathcal{L}$ be a family of languages effectively closed under inverse homomorphism and intersection with a regular set. Assume that the languages in $\mathcal{L}$ have effectively constructible modular Parikh maps. Then there is an algorithm to determine given a language L in $\mathcal{L}$ and a 1-way nondeterministic multicounter machine M, where each counter makes at most one reversal (i.e. once a counter is decremented, it can no longer be incremented), whether $L \cap T(M)$ is empty. Moreover, $L \cap T(M)$ has an effectively constructible modular Parikh map.

Proof: The proof is essentially the one given in [8] with the word "semilinear" replaced by "modular". Since some details of the proof will be used in the subsequent discussion we include a sketch here. Ibarra augments the input alphabet, Σ, of M with special symbols indicating incrementing and decrementing for each of the counters in M, and symbols representing each transition of M. He constructs over this alphabet a regular set, R_M, consisting of strings representing sequences of transitions leading from the start state of M to an accepting state. Each segment of such a string, representing one move, contains information encoding the changes in the counters, the top symbols of the counters, the states, and the input character for the move. This alphabet is called Δ. For details of the construction, see [8].

Ibarra defines a homomorphism, h, from Δ^* to Σ^* by having h erase all letters except those in Σ, which are left alone. This means that the inverse image of a word in Σ^* describes a possible accepting computation for that word in M. By the assumed closure properties, $L' = h^{-1}(L) \cap R_M$ is in $\mathcal{L}$. Hence $Q_1 = f_\beta(L')$ is an effectively constructible modular set. Here β is the alphabet Δ, arranged with the characters from Σ first, followed by the characters representing counter increments and decrements, in pairs, followed by the other characters of Δ.

If we let Q_2 be the set of vectors of the same length as those in Q_1, but require the components to be equal, in pairs, at the places corresponding to the incrementing and decrementing characters, then $Q_3 = Q_1 \cap Q_2$ is an effectively constructible modular set of vectors that correspond to accepting computations in M. This is because words accepted by M are assumed to cause all the counters to end up at zero, and hence have exactly as many decrements as increments.

---

[2] We follow each theorem from [8] with its designation in that source, inside parentheses.

Deleting the last 2k+s coordinates of the vectors in Q_3, gives the effectively constructed modular set Q_4, which is the Parikh map of L∩T(M). The result follows from the decidability of the emptiness problem for Q_4.□

Before his second theorem, Ibarra proves a lemma (his Lemma 1) that every 1-way nondeterministic machine with k 1-reversal counters can be made equivalent to a similar machine in *normal form*. Briefly, this means that: all counters are zero upon acceptance; each move changes some counter; and, during any given computation, all counters attain the same positive value (but the value may be different for different computations).

There seems to be an error in the proof of Theorem 2 in [8]. Modularity or semilinearity are not involved here, so the only change in the statement of this theorem is the addition of the requirement that the family of languages be closed under post-concatenation with a marked regular set, which serves to correct this apparent error. (Note that almost all well known families of languages satisfy this property.)

Theorem 4.2(Theorem 2): Let $\mathcal{L}$ be a family of languages effectively closed under inverse homomorphism, intersection with a regular set, and post-concatenation with a marked regular set. Then (1) is decidable if and only if (2) is decidable:

(1) Given a language, L, in $\mathcal{L}$ and a 1-way nondeterministic multicounter machine, M, where each counter makes at most one reversal, is the set L∩T(M) empty?

(2) Given a language, L, in $\mathcal{L}$ over the alphabet Σ, is the set:

$$E(L)=\{x|x \text{ is in L and for all a,b in } \Sigma, \text{ the number of a's in x=the number of b's in x }\}$$

empty?

Proof: We sketch the proof in [8](see also [9]) to see where the additional hypothesis seems necessary. Clearly (2) is decidable if (1) is; just make a machine, M, with a counter for each letter in Σ, and let it count the letters and test that they occur equally often in a given string.

The converse is much harder. Ibarra makes a construction similar to that in the previous proof, except for two things. Where before a single $\$_j$ was used to indicate that the jth counter was decremented, now $(\$_j)^{2k}$ is used; and the same for $\dagger_j$. The purpose of this duplication is to ensure that, in the model of an accepting computation being built, the †'s and $'s appear more often than the other characters.

The second difference is that now R_M consists of strings like this:

$$\alpha_1\alpha_2\ldots\alpha_m w$$

where w is any string in $(\Delta-\{\dagger_i,\$_i|1\leq i\leq k\})^*$. The purpose of the trailing strings, w, is to allow padding with enough characters *other than the various †'s and $'s* to make all characters in the resulting string equally numerous.

The difficulty in the original proof occurs here. If we let $L'=h^{-1}(L)\cap R_M$, as before, and test E(L') for emptiness, then, suppose we find there is some y in E(L'). That means h(y) is in L. Now, the string, y, in L' looked like this:

$$y = \alpha_1\alpha_2 \ldots \alpha_m w,$$

where the α's are move-strings. We know h(y) is in L, and the series of move-strings describes an accepting computation for M; but the word described by the move-strings is not, in general, h(y). This is because there will usually have to be letters from Σ in the added string w, to make them be equally numerous with the †'s and \$'s. So h(w) is not empty. Thus one cannot conclude from this argument that $L\cap T(M)$ is nonempty, just because E(L') is.

We can correct the proof, given the new hypothesis, as follows. Keep the new duplication of the †'s and \$'s, but delete the trailing string, w. Let R_2 be the marked regular set:

$$\#(\#)^*(\Delta - \{\dagger_i,\$_i \mid 1\leq i\leq k\})^*$$

where "#" is the marking character. Then let $L'=(h^{-1}(L)\cap R_M)R_2$. Now, if there is something in $L\cap T(M)$, its accepting computation will be in $h^{-1}(L)\cap R_M$, and padding it with the proper string will make all the characters be equally numerous. Thus there will be something in E(L').

Conversely, if there is something in E(L'), then the part of it that is in $h^{-1}(L)\cap R_M$ must correspond to something in $L\cap T(M)$ when mapped back into L by the homomorphism h. Thus decidability of (2) implies decidability of (1) when one has the extra hypothesis given.□

We proceed to point out, as Ibarra does that properties (1) and (2) of Theorem 4.2 are decidable if the languages in $\mathcal{L}$ have effectively constructible modular Parikh maps.

We now observe that the class of languages $\mathcal{L}_1$ described in Chapter 2 satisfies the hypotheses of Theorems 4.1 and 4.2. (Using standard techniques the reader can easily show that $\mathcal{L}_1$ is effectively closed under: (1) inverse homomorphisms, (2) intersection with a regular set, and (3) post-concatenation with a marked regular set.) Thus, the solution techniques described in [8] apply to this class of languages.

The remaining results in [8] remain valid with the new hypotheses except when the infiniteness problem is involved instead of emptiness. For example, since the proof depends on finiteness instead of emptiness, we cannot prove a modular version of Ibarra's Theorem 6, but Theorem 7 does generalize. As a result, if $\mathcal{L}$ is a family of languages having effectively constructible modular Parikh maps, which is effectively closed under inverse homomorphism and intersection with a regular set, then:

1. There exists an algorithm to decide given an arbitrary language in $\mathcal{L}$ and a language accepted by a one-way nondeterministic reversal-bounded multicounter machine whether their intersection is empty.

2. There is an algorithm to decide given a language $L \in \mathcal{L}$ and two 2-way deterministic sequential transducers (2DST's) whether they are equivalent on L.

The concept of modularity seems to offer a new tool in the study of formal languages, allowing some previously-established results to be extended to new classes of languages and accepting machines. We are left at this point with four principal open questions whose solutions, if forthcoming, should allow even wider applications of these techniques.

Problem 4.1: Show that the finiteness problem is decidable for modular sets.

A major aspect that makes the solution sets of Presburger formulas such a powerful tool is their internal characterization as semilinear sets of vectors. Thus we have:

Problem 4.2: Find an analogous internal characterization for modular sets.

Additional, more general, open questions of interest are the following.

Problem 4.3: What other language classes have effectively constructible modular Parikh maps?

Problem 4.4: What other properties of languages with semilinear Parikh maps also hold for classes with modular Parikh maps?

Acknowledgement: We would like to thank Professor Oscar H. Ibarra for suggestions which improved the presentation of our results.

Bibliography

[1] Chomsky, N., "Three models for the description of language," *IRE Transactions on Information Theory,* Vol. IT2, 1956, pp. 113-124.

[2] Ginsburg, S. and Spanier, E., "Bounded Algol-like languages," *Trans. Amer. Math. Soc.,* Vol. 113, 1964, pp. 333-368.

[3] Greibach, S.A., "One-way finite visit automata," *Theoretical Computer Science,* Vol. 6, 1978, pp. 175-221.

[4] Gurari, E.M., and Ibarra, O.H., "Two-way counter machines and Diophantine equations," *Jour. ACM,* Vol. 29, No. 3, 1982, pp. 863-873.

[5] Gurari, E.M., and Ibarra, O.H., "The complexity of decision problems for finite-turn multicounter machines," *Jour. Comput. Syst. Sci.,* Vol. 22, 1981, pp. 220-229.

[6] Gurari, E.M., "Two-way counter machines and finite-state transducers," Technical Report, Ohio State University, Department of Computer and Information Science.

[7] Harrison, M.A., *Introduction to Formal Language Theory,* Addison-Wesley Publishing Co., Reading, Mass., 1978.

[8] Ibarra, O.H., "2DST mappings on languages and related problems," *Theoretical Computer Science,* Vol. 19, 1982, pp. 219-227.

[9] Ibarra, O.H., *Some decision problems and applications,* Technical Report 81-21, University of Minnesota, 1981.

[10] Ibarra, O.H., "Reversal-bounded multicounter machines and their decision problems," *Jour. ACM,* Vol. 25, 1978, pp. 116-133.

[11] Ibarra, O.H., "Simple matrix languages," *Information and Control,* Vol. 17, 1970, pp. 359-394.

[12] Ibarra, O.H., "Controlled pushdown automata," *Information Sci.,* Vol. 6, 1973, pp. 327-342.

[13] Kasai, T., "An hierarchy between context-free and context-sensitive languages," *Jour. Comput. System Sci.,* Vol. 4, 1970, pp. 153-160.

[14] Lipschitz, L., "The Diophantine problem for addition and divisibility," *Trans. Amer. Math. Soc.,* Vol. 235, 1978, pp. 271-283.

[15] Matijasevic, Y., "Enumerable sets are Diophantine," *Soviet Math. Doklady,* Vol. 11, 1970, pp. 354-357.

[16] Parikh, R.J., "On context-free languages," *Jour. ACM,* Vol. 13, 1966, pp. 570-581.

[17] Rabin, M.O. and Scott, D., "Finite automata and their decision problems," *IBM Journal of Res. & Dev.,* Vol. 3, 1959, pp. 114-125.

[18] Rajlich, V., "Absolutely parallel grammars and two-way finite state transducers," *Jour. Comput. System Sci.,* Vol. 6, 1972, pp. 324-342.

[19] Robinson, J., "Definability and Decision Problems in Arithmetic," *Jour. Symb. Logic,* Vol. 14, 1949, pp. 98-114.

[20] Rosenkrantz, D. J., "Programmed grammars and classes of formal languages," *Jour. ACM,* Vol. 16, 1969, pp. 107-131.

[21] Siromoney, R., "Finite-turn checking automata," *Jour. Comput. System Sci.,* Vol. 5, 1971, pp. 549-559.

[22] Turing, A.M., "On computable numbers, with an application to the Entscheidungs problem," *Proc. London Math. Soc.,* Vol. 2-42, 1936, pp. 230-265.