An Introduction to Computable Model Theory on Groups and Fields

Russell Miller[*]

January 17, 2011

Abstract

We introduce the standard computable-model-theoretic concepts of a computable group and a computable field, and use them to illustrate the sorts of questions about groups and fields which computability theorists investigate. This article is intended for group theorists with some background in algorithmic questions, such as the undecidability of the word problem and the conjugacy problem for finitely presented groups.

1 Introduction

The undecidability of the word problem for finitely presented groups is widely known among group theorists. This result, established independently by Boone and by Novikov in the 1950's (see [1] and [16]), answers a natural computability-theoretic question about groups, and is frequently cited (by computability theorists, as evidence of the relevance of their field of study) and used (by group theorists, especially recently, as interest in the use of groups to construct cryptosystems has increased). Related problems have also been shown to be undecidable, notably the conjugacy problem and the

[*]The author was partially supported by National Science Foundation grant #DMS-1001306, by grant #13397 from the Templeton Foundation, and by grants #62632-00 40 and #63286-00 41 from The City University of New York PSC-CUNY Research Award Program.

isomorphism problem for finitely presented groups. On the other hand, certain more specific instances of the word problem have been proven decidable. Decidability questions continue to be of significant interest for many group theorists.

Computable model theory represents a more general framework for such questions. It is the hybridization of two of the principal branches of mathematical logic: *model theory*, which studies the relationship between semantics (actual mathematic structures) and syntax (the formulas we write down to describe those structures), and *computability theory*, also known as *recursion theory*, which studies our ability to compute various functions on the natural numbers $\mathbb{N}$ and ranks such functions according to the difficulty of computing them. Therefore, computable model theory investigates the feasibility of deciding facts or computing functions on mathematical structures. This article, aimed at group theorists having interest in issues of decidability, is written to introduce the basic definitions used in computable model theory, and to give a sample of the sort of results which follow from these concepts. Instead of applying computability to model theory in general, however, we will restrict ourselves to computable groups and computable fields, the latter being the principal area of expertise of the author. By the end of the article, we will even be able to present a few new results and explain how they follow from work in the recent paper [13].

2 Basic Definitions

Model theorists normally discuss groups in the context of a language with an equality symbol and a single binary operation $\cdot$, and fields in the context of a language with equality and two distinct such operations $\cdot$ and $+$. In each case, we can express the relevant axioms (for groups, or for fields) in this language, using variables and the standard logical connectives (and, or, and negation, say), along with quantifiers $\exists$ and $\forall$ over the elements of the group or the field. A *structure* in the language of groups just consists of a set S and a binary function from $S \times S$ to S, and a group is then defined as a structure in this language which satisfies those group axioms when the binary function is interpreted as the $\cdot$ operation. The same applies to fields, with two binary functions on the underlying set S. Statements (or *formulas*) about groups and fields can likewise be built from these languages, and one can ask whether a specific group satisfies a specific formula, or whether all groups satisfy that formula, and so on.

In computable model theory, we assume that the binary operation(s) of our group or field are given by computable functions (as defined below in Section 3). Definition 2.1 makes this precise.

Definition 2.1 A *computable group* G consists of a set $\{a_0, a_1, a_2, \ldots\}$, indexed by elements of $\mathbb{N}$ or by elements of a finite subset of $\mathbb{N}$, with a binary operation $\cdot$ on these elements such that:

- $\{a_0, a_1, a_2, \ldots\}$ forms a group under this operation, and
- there exists a computable function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all elements a_i and a_j,
$$a_i \cdot a_j = a_{f(i,j)}.$$

A *computable field* consists of a set $\{x_0, x_1, x_2, \ldots\}$, indexed likewise, which forms a field under two operations $+$ and $\cdot$ which are similarly given by two computable functions g and h:

$$x_i + x_j = x_{g(i,j)} \qquad x_i \cdot x_j = x_{h(i,j)}.$$

To help keep groups and fields distinct in the reader's mind, we will maintain here a convention (otherwise unknown) of using a_i for elements of computable groups and x_i for elements of computable fields. Of course, it is quite possible to use $\mathbb{N}$ itself as the underlying set, rather than naming elements a_i or x_j, but it creates havoc, since the element x_0 need not be the additive identity element (usually denoted by 0) of a computable field. Indeed, we will sometimes want to refer to other elements of a computable field, such as 5, and this will mean the rational prime 5 in the field, i.e. the sum $(1+1+1+1+1)$ of the multiplicative identity with itself five times.

It is quickly seen from Definition 2.1 that the other basic operations on these structures are also computable. One can find the identity element in a computable group by computing $f(0,0), f(1,1), \ldots$, since the identity element a_i has the unique index i satisfying $f(i,i) = i$. With this, one can also then compute the inversion function (that is, the function v such that $a_n^{-1} = a_{v(n)}$), just by computing $f(n,0), f(n,1), \ldots$ until the unique m with $f(n,m) = i$ is found. Likewise, in a field, one can compute both inversion functions (checking, before searching for the reciprocal, to make sure that the input is not the element 0), and therefore can also compute subtraction and division.

Definition 2.1 requires that we index elements of the group or field using $\mathbb{N}$, and thus rules out any uncountable groups or fields. (Allowing finite subsets of $\mathbb{N}$ is our way of allowing finite groups and fields into the picture.) This is mostly inevitable, since the standard notion of computability – whether given using Turing machines as in Section 3, or using register machines, recursive functions, or what you will – only applies to functions on the natural numbers $\mathbb{N}$. This notion is rigorously defined in many texts; the canonical computability text for graduate study is [19]. We will say that a set $S \subseteq \mathbb{N}$ is *computable*, or *decidable*, when its characteristic function $\chi_S : \mathbb{N} \to \mathbb{N}$ is computable. The terms *computable* and *decidable* are mostly interchangeable in this context. To the extent that there is a convention, functions are called computable (or else noncomputable) and sets are called decidable (or else undecidable).

Examples of some computable groups and fields will be given in Section 4. Having omitted so far to give any specifics about computable functions, we will first introduce a few computability-theoretic preliminaries. Much of this information (some in expanded form) can also be found in the expository works [12] and [14], both of which go into further detail about computable fields. Other notable articles about computable fields, more at the research level, include [3], [4], [5], [11], [17], and [20].

3 Computability Background

One of the attractions of the standard Turing model computation on $\mathbb{N}$ is that so many distinct definitions, all reasonable in various intuitive respects, give rise to the exact same class of functions: the so-called computable functions. Rigorous definitions can be found in any standard text on computability, including [9], [18], and [19]. For our purposes, a *Turing machine* is an ordinary computer, operating according to a finite program, which accepts a natural number as input and runs its program in discrete steps on that input. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *computable* if there is a Turing machine which computes f. Specifically, when running on each input $n \in \mathbb{N}$, the program should halt within finitely many steps and give output $f(n)$. There is no bound on the amount of memory used, except that only one bit can be accessed or written over at each single step.

More generally, we consider *partial functions* $\varphi : \mathbb{N} \to \mathbb{N}$, for which (despite the similarity of notation) the domain is allowed to be any subset of

$\mathbb{N}$. The possibility that a program never halts makes the partial functions the natural class for our definitions, since every Turing machine computes some partial function, namely that φ whose domain is the set of inputs on which the machine eventually halts, with $\varphi(n)$ being the output of the machine for each such input n. We write $\varphi(n)\downarrow$, and say that $\varphi(n)$ *converges*, if $n \in \mathrm{dom}(\varphi)$; otherwise $\varphi(n)$ *diverges*, written $\varphi(n)\uparrow$. Since the program for a Turing machine is required to be finite, we have only countably many programs in all. Hence only countable many partial functions are computable, with the remaining (uncountably many) ones being noncomputable. It is not hard to define a noncomputable function: just imitate Cantor's diagonal proof that $\mathbb{R}$ is uncountable.

The concept of *computable enumerability* is essential to computability theory. A set $S \subseteq \mathbb{N}$ is *computably enumerable*, abbreviated *c.e.*, if it is empty or is the range of some computable function f with domain $\mathbb{N}$. Intuitively, this says that there is a mechanical way to list out the elements of S: just compute $f(0)$, then $f(1)$, etc., and write each one on our list. Computably enumerable sets are "semi-decidable," in the following sense.

Fact 3.1 *A subset $S \subseteq \mathbb{N}$ is decidable (i.e. χ_S is computable) iff both S and its complement $\overline{S}$ are computably enumerable.*

Fact 3.2 *For any set $S \subseteq \mathbb{N}$, the following are equivalent:*

- *S is computably enumerable.*
- *S is the range of an injective partial computable function.*
- *There is a decidable set $R \subseteq \mathbb{N} \times \mathbb{N}$ with $S = \{x : \exists y\ (x, y) \in R\}$.*
- *S is the domain of a partial computable function.*

For the R in this fact, we need to consider subsets of Cartesian products $\mathbb{N}^k$ as well. For this we use the function

$$\beta_2(x, y) = \frac{1}{2}(x^2 + y^2 + x + 2xy + 3y),$$

which is a bijection from $\mathbb{N}^2$ onto $\mathbb{N}$. (In checking this, bear in mind that for us 0 is a natural number.) To justify our view of β_2 as a computable function, notice that if π_1 and π_2 are projections, then both functions $\pi_i \circ \beta_2^{-1}$ are computable, and so, for any given x and y, we can search through all possible outputs $n \in \mathbb{N}$ until we find one for which $\pi_1(\beta_2^{-1}(n)) = x$ and $\pi_2(\beta_2^{-1}(n)) = y$. So we may use this bijection β_2 to treat $\mathbb{N}^2$ as though it were just $\mathbb{N}$, and then define $\beta_3(x, y, z) = \beta_2(x, \beta_2(y, z))$ and so on. Indeed, the bijection β defined by

$$\beta(x_0, \ldots, x_k) = \beta_2(k, \beta_{k+1}(x_0, \ldots, x_k))$$

maps the set $\mathbb{N}^*$ of all finite sequences of natural numbers bijectively onto $\mathbb{N}$. (For a computable field F, this gives us a way of allowing polynomials from $F[X]$ to be the inputs to a computable function.)

Now that we know how to handle tuples from $\mathbb{N}$ as inputs, we can see that there is a *universal Turing machine*. The set of all possible programs is not only countable, but can be coded bijectively into the natural numbers in such a way that a Turing machine can accept an input $e \in \mathbb{N}$, decode e to figure out the program it coded, and run that program. The universal Turing machine accepts a pair (e, x) as input, decodes e into a program, and runs that program on the input x. It defines a partial computable function $\varphi : \mathbb{N}^2 \to \mathbb{N}$ which can imitate every partial computable function ψ: just fix the correct e, and we have $\psi(x) = \varphi(e, x)$ for every $x \in \mathbb{N}$ (and with $\psi(x)\uparrow$ iff $\varphi(e, x)\uparrow$, moreover). This enables us to give a computable list of all partial computable functions, which we usually write as $\varphi_0, \varphi_1, \ldots$, with $\varphi_e(x) = \varphi(e, x)$ for a fixed universal partial computable function φ. In contrast, there is no computable list of all the total computable functions (i.e. those with domain $\mathbb{N}$); if there were, one could use it to diagonalize and get a new total computable function not on the list!

Likewise, using Fact 3.2, this yields a computable list of all computably enumerable sets $W_0, W_1, \ldots$, with $W_e = \mathrm{dom}(\varphi_e)$. We can view them as the rows of the universal c.e. set $W = \mathrm{dom}(\varphi) = \{(e, x) : \varphi_e(x)\downarrow\} \subseteq \mathbb{N}^2$. On the other hand, there is no such computable listing of all decidable sets.

The principal remaining fact we will need is simply stated.

Fact 3.3 *There exists a c.e. set which is not decidable.*

A simple definition of one such is

$$K = \{e : \varphi_e(e)\downarrow= 0\}.$$

The idea is that φ_e cannot equal χ_K, because $e \in K$ iff $\varphi_e(e) = 0$ iff φ_e guesses that e is *not* in K. Of course, if $\varphi_e(e)$ never converges, we never add

e to K, and again we see that $\chi_K \neq \varphi_e$, simply because $e \in \mathrm{dom}(\chi_K)$. For reasons explained in Section 6, we often denote this set K by $\emptyset'$.

Many similar sets can be defined; the famous one is the domain of the function φ computed by the universal Turing machine. This set is usually called the *halting problem*, since it tells you exactly which programs converge on exactly which inputs. If it were decidable, then we could use it to compute χ_K, which is impossible. Indeed, if $\mathrm{dom}(\varphi)$ were decidable, then every c.e. set would be decidable. Since one can readily produce a computable (total) bijection from $\mathbb{N}$ onto $\mathbb{N}$ mapping the halting problem onto $\emptyset'$, the set $\emptyset'$ itself is sometimes also called the halting problem.

4 Computable Groups and Fields

As a first example for groups, we use the free (nonabelian) group $\mathbf{FG}_\omega$ on countably many generators. To build a computable group G of this isomorphism type, with domain $\{a_0, a_1, a_2, \ldots\}$, we will use the bijection $\beta : \mathbb{N}^* \to \mathbb{N}$ defined in the previous section, and also a bijection $\gamma : \mathbb{N} \to \mathbb{Z}$ with $\gamma(k) = (-1)^k \cdot \lfloor \frac{k+1}{2} \rfloor$. The idea is that, if we call the generators $z_0, z_1, \ldots$, and $\beta(k_0, \ldots, k_n) = i$, then the group element a_i represents the word

$$z_{|\gamma(k_0)|}^{(-1)^{k_0}} \cdots z_{|\gamma(k_n)|}^{(-1)^{k_n}}.$$

So the generators are really $a_{\beta(0)}, a_{\beta(2)}, a_{\beta(4)}, \ldots$, with $a_{\beta(2m+1)} = a_{\beta(2m)}^{-1}$. The group multiplication on G is given by concatenation

$$a_{\beta(j_0,\ldots,j_m)} \cdot a_{\beta(k_0,\ldots,k_n)} = a_{\beta(j_0,\ldots,j_m,k_0,\ldots,k_n)},$$

except that if $k_0 = j_m + (-1)^{j_m}$, then the two middle elements cancel each other and disappear, and then similarly for the next two. A moment's thought should convince the reader that this multiplication is a computable function f on the indices, so that this G really is a computable group, isomorphic to the free group on the generators $a_{\beta(0)}, a_{\beta(2)}, a_{\beta(4)}, \ldots$, exactly as claimed. The point is not really that the particular group $\mathbf{FG}_\omega$ has a free presentation, of course, but rather that this is a straightforward way of turning $\mathbf{FG}_\omega$ into a computable group: the process is best imagined as programming a computer to use natural numbers as codes for words in the generators and to multiply such words (i.e. their codes) together. We sometimes refer to this group G as a *computable presentation* of $\mathbf{FG}_\omega$, or a *computable copy* of $\mathbf{FG}_\omega$; this simply means a computable group isomorphic to $\mathbf{FG}_\omega$.

Likewise, there is a computable field F isomorphic to the field $\mathbb{Q}$ of rational numbers. For this we wish to think of each x_i as a fraction with integer numerator and natural-number denominator, without letting our domain repeat any fractions. Since we have the computable bijection β_2 from Section 3, we can define $h(0) = 2 = \beta_2(0,1)$ and $h(n+1)$ to be the least $k > h(n)$ for which $\pi_1(\beta_2^{-1}(k))$ is relatively prime to $\pi_2(\beta_2^{-1}(k))$. This allows us to define computable functions $\texttt{num}(2n) = \pi_1(\beta_2^{-1}(h(n)))$ and $\texttt{denom}(2n) = \pi_2(\beta_2^{-1}(h(n)))$ for all $n \in \mathbb{N}$, giving the numerators and denominators of the field elements $x_0, x_2, x_4, \ldots$. We treat x_{2n+1} as the negative of x_{2n+2}, and define addition and multiplication on the domain $\{x_0, x_1, x_2, \ldots\}$ by doing arithmetic on fractions.

The reader is invited to attempt to build computable fields of isomorphism types such as $\mathbb{Q}(X)$, $\mathbb{Q}(\sqrt{2})$, $\mathbb{Q}(X_1, X_2, \ldots)$, and other well-known countable fields and groups. The p^n-element field $\mathbb{F}_{p^n}$ also has a computable presentation, of course, as does every finite group. Moreover, every finitely generated field extension of any computable field also has a computable presentation. (We will see below that for groups, the analogous statement is false!)

Noncomputable groups and fields, in the naive sense, are easy to come by. First, of course, any group with a different underlying set than $\{a_0, a_1, \ldots\}$ is technically not a computable group. But even groups with this domain can easily fail to be computable. We now prove the existence of noncomputable groups with domain $\{a_0, a_1, \ldots\}$ which are isomorphic to $\mathbf{FG}_\omega$. Let p be any bijection from $\mathbb{N}$ onto $\mathbb{N}$, and notice that then $\{a_0, a_1, \ldots\}$ forms a group isomorphic to $\mathbf{FG}_\omega$ under the following multiplication:

$$a_i \cdot a_j = a_{p^{-1}(f(p(i),p(j))}$$

where f was the function in the computable presentation G of $\mathbf{FG}_\omega$ given above. This can be thought of as "pulling G back via p^{-1}." It is possible that this pullback might still be a computable group, even if p is not a computable bijection. However, there are continuum many such bijections p, each of which pulls back G to a group distinct from all the others (though isomorphic to them all). Since only countably many of these continuum many distinct groups could possibly be computable, we see that there must exist many noncomputable presentations of $\mathbf{FG}_\omega$ on the domain $\{a_0, a_1, \ldots\}$.

For a more concrete example of a noncomputable group H, take the computable presentation of $\mathbf{FG}_\omega$ above, and change the definitions to make the generators $a_{\beta(0)}$ and $a_{\beta(2n)}$ commute with each other iff n lies in the halting problem $\emptyset'$. Of course, this group H would no longer be isomorphic to $\mathbf{FG}_\omega$, but the point is that it would also not be computable, since, if g is the function on indices defined by its multiplication, then

$$(\forall n \in \mathbb{N})[n \in \emptyset' \iff g(0, 2n) = g(2n, 0)],$$

so that the ability to compute g would allow us to decide membership in $\emptyset'$, which is impossible.

However, this group H is isomorphic to a computable group. We can think of it as a free group which is nonabelian except that one particular generator commutes with half of the other generators (but not with the other half of them). By now the reader should be able to imagine how to create a computable presentation of that group. Our next project, therefore, for which we turn to fields, is to create a countable field F such that no computable field can be isomorphic to F.

Consider again the noncomputable c.e. set $\emptyset'$ from Fact 3.3. Write p_n for the n-th prime number (so $p_0 = 2$, $p_4 = 11$, etc.), and let $F_{\overline{\emptyset'}}$ be the following field extension of $\mathbb{Q}$:

$$F_{\overline{\emptyset'}} = \mathbb{Q}[\sqrt{p_n} \ : \ n \notin \emptyset'].$$

Now the set of primes is computable, and so in any field of characteristic 0, it is easy to list out the prime numbers $p_0, p_1, \ldots$, just by adding 1 to itself. (Specifically, the function v with $p_n = a_{v(n)}$ is computable.) If F were a computable field isomorphic to $F_{\overline{\emptyset'}}$, then the following process would contradict the noncomputability of $\emptyset'$. Each time a field element appears in F whose square equals p_n for any n, enumerate that n into a set W. By the definition of $F_{\overline{\emptyset'}}$, this W would equal the complement $\overline{\emptyset'}$, and we would have a computable enumeration of this complement, which is impossible, by Fact 3.1.

In light of this, it may seem surprising that the field $F_{\emptyset'} = \mathbb{Q}[\sqrt{p_n} \ : \ n \in \emptyset']$ is computably presentable. Yet it is: we will build a computable presentation of $F_{\emptyset'}$. To "build" a field usually means that we give finitely much of it at a time, first defining the addition and multiplication functions only on the domain $\{x_0, \ldots, x_j\}$, then extending them to $\{x_0, \ldots, x_{j+1}\}$, and so on. We do this according to an algorithm, and if we wish later to compute the field addition or multiplication, we simply run this same algorithm until it defines the sum or product we seek. (Of course, our algorithm must decide within finitely much time which x_k is to be the sum $x_i + x_j$; and once it has decided, it may not change its mind!)

So we start building a computable presentation of $\mathbb{Q}$, similar to the one given above, and simultaneously start enumerating $\emptyset'$. (Think of a timesharing procedure, allowing us to do both these processes at once.) Whenever a new element n appears in $\emptyset'$, we continue building our field until the element p_n has appeared in it, then stop for long enough to define the multiplication so that the next new element is the square root of p_n. Then we continue building the field, always treating this new element as the square root of p_n when defining the addition and multiplication. Since every $n \in \emptyset'$ eventually appears in our enumeration, this builds a computable field isomorphic to $F_{\emptyset'}$.

The key here is that the statement "p_n has a square root" is an existential formula, with free variable n:

$$(\exists x)[x \cdot x = (1 + 1 + \cdots + 1) \quad (p_n \text{ times})].$$

The statement within the square brackets defines a computable set of elements x. Therefore, in a computable field, the set of numbers n satisfying this existential formula is a computably enumerable set, by Fact 3.2. $\emptyset'$ itself is c.e., so having this set equal $\emptyset'$ (as in $F_{\emptyset'}$) is not a problem. Indeed, we could build a similar field F_W for any c.e. set W. However, since the complement $\overline{\emptyset'}$ is not c.e., the field $F_{\overline{\emptyset'}}$ is not computably presentable.

There are similar ways of constructing a countable group with no computable presentation. However, considering our audience, we will instead appeal to a known result: the existence of a finitely presented group G with undecidable word problem. Fix any such group G, and suppose that H were a group isomorphic to G, but with domain $\{a_0, a_1, \ldots\}$. Then H would have a finite set of generators $a_{n_0}, \ldots, a_{n_k}$, the images of the generators of G under the isomorphism. Also, we may fix the i such that a_i is the identity element of H. Now, for any word in the generators of G, we could take the corresponding word in the generators of H and use the multiplication function h in H to check whether this word is the identity. (For instance, for the word $a_{n_2}a_{n_7}a_{n_2}$, just check whether $h(n_2, h(n_7, n_2)) = i$.) If the multiplication function were computable, this would decide the word problem for G, which is impossible; so there can be no computable presentation of G. The important point here is that the generating set for G was finite, which allowed us to conclude, without even knowing what the indices $n_0, \ldots, n_k$ actually were, that the map taking the j-th generator of G to a_{n_j} was computable: after all, every finite function is computable. Then, since the definition of computable group immediately implies that we can decide whether any two group elements (named by their indices) are equal, we were able to decide whether an arbitrary word equals the identity. With computable groups, we took the equality relation to be decidable, by definition, which is a different approach than the notion of computation on groups which gave rise to the word problem.

The word problem can be expressed in the context of computable groups, however. If a finitely presented group G has k generators and has relators $R_1, \ldots, R_m$, then G is isomorphic to the quotient of the free group H on k generators by the subgroup J of H generated by the relators (viewed as words in the appropriate generators of H). H can readily be given a computable presentation, and J is also isomorphic to a computable group. Moreover, the subgroup J is computably enumerable within H: to list out the elements of J, just list the relators, their inverses, and all finite products thereof. Again, since there are only finitely many relators, this is a computable process. However, the subset J is not necessarily decidable; indeed membership in J is precisely the word problem for the original group G, which is isomorphic to the quotient group H/J. The content of the Boone-Novikov Theorem is the statement that a finitely generated subgroup of a computable, finitely generated free group need not be decidable.

We remark here (without proof) that, in contrast to the word problem, the conjugacy problem *can* be undecidable for a computable group. It is expressed by an existential formula

$$\langle x, y\rangle \in (\text{Conjugacy Problem}) \iff (\exists z \in G)\ zxz^{-1} = y$$

and therefore the set of such pairs $\langle x, y\rangle$ is always a computably enumerable set, but nothing about computable groups forces this c.e. set to be decidable.

5 Bases for Computable Groups and Fields

We will now investigate the concept of a basis, both for groups G (a maximal independent subset of G) and for fields (a transcendence basis). This subject will illustrate the ways in which computability theorists discuss the relative difficulty of deciding membership in two different sets. Of course, if the two sets are both undecidable, then in some sense they are "equally" difficult (namely, impossible) to decide. However, there is a stronger sense in which a set $A \subseteq \mathbb{N}$ and its complement $\overline{A}$ in $\mathbb{N}$ are equally difficult: any decision procedure for A clearly would give us a decision procedure for $\overline{A}$ as well, and vice versa. Of course, if A is undecidable, then no such decision procedure exists, but there is still a strong intuition that A and $\overline{A}$ are equally difficult to decide, in a way in which two randomly chosen subsets of $\mathbb{N}$ almost certainly would not be. This is formalized in the notion of *oracle Turing computation*, which we discuss below.

Moving to concrete topics, we first ask, for a computable group G and an arbitrary finite tuple $(a_{n_0}, \ldots, a_{n_k})$ of elements of G, how difficult it is to decide which $m \in \mathbb{N}$ have the property that a_m lies in the span of $(a_{n_0}, \ldots, a_{n_k})$ within G. Of course, the difficulty may depend on the tuple: if $(a_{n_0}, \ldots, a_{n_k})$ happen to generate G, then it is extremely easy to decide this question, whereas we saw above (discussing the Word Problem) that for some computable groups, the same problem can be undecidable. What one would really like, of course, would be a single procedure which accepts both m and $(n_0, \ldots, n_k)$ (for any length k) as inputs, and gives the correct answer. If this happened, we would say that the span problem is *uniformly decidable* for G. Similar notions apply to computable fields, asking about membership in the subfield of F generated by a finite subset of F, and it is worth noting that in contrast to groups, every finitely generated computable field has uniformly decidable span problem. (This result goes back essentially to Kronecker, in the paper [8] from 1882.)

For both groups and fields, the span problem is often adjusted to ask about independence results. In field theory, the notion of *algebraicity* is paramount: x is algebraic over a subfield $E \subseteq F$ iff x is a root of some nonzero polynomial in $E[X]$. (If E is generated by $e_0, \ldots, e_k$, then we will speak of x being algebraic over $e_0, \ldots, e_k$.) Likewise, a group element $a \in G$ is *dependent* on a subgroup $H \subseteq G$ if there is some nontrivial word in a and finitely many elements of H which equals the identity a_i. (Here *nontrivial* means roughly that it is not just the elements of H which make the relation equal to a_i.) For instance, for an element b of a free group, not only are b^2, b^3, and b^{-1} dependent on (the subgroup generated by) b, but also every a with $a^n = b^m$ for some $n > 0$ and $m \in \mathbb{Z}$ is dependent on b. Torsion elements of a group are always dependent, even on the identity subgroup, and likewise any field element algebraic over the prime subfield (for instance, a square root of 2, should the field have one) is algebraic over every subfield.

A first question about a computable group or computable field, therefore, is whether the set of torsion elements (resp. algebraic elements) is decidable. In general, the answer is negative. These sets are always computably enumerable: in a field F of characteristic 0, for example, one can enumerate the prime subfield Q of F (which is just a copy of the rationals), enumerate all polynomials $q(X) \in Q[X]$, and start plugging various field elements x systematically into each, multiplying and adding to find the value $q(x)$ and checking whether it equals 0. Any time $q(x)$ (for any q and x) does equal 0, we enumerate x onto our list of algebraic elements, and this process lists out all elements of F algebraic over Q. (Alternatively, notice that the definition of algebraicity can be expressed with an existential quantifier, and apply Fact 3.2.) However, there may be no analogous way of listing out the elements of F transcendental (i.e. not algebraic) over Q, and thus, by Fact 3.1, the algebraic elements may form an undecidable set.

From here, one proceeds naturally to questions about transcendence bases for fields F, i.e. subsets of F which are maximal with the property that no nontrivial polynomial relation (over the prime subfield) holds on any finite subset. Since computable fields are always countable, we can think of a transcendence basis $B = \{b_0, b_1, \ldots\}$ as a set with the property that no b_{n+1} is algebraic over $\{b_0, \ldots, b_n\}$ and that F is algebraic over the subfield generated by B. Of course, a field generally has many distinct transcendence bases, and some may have very high complexity. However, it is well known that a computable field must have a transcendence basis satisfying the following definition.

Definition 5.1 A set $S \subseteq \mathbb{N}$ is *limit-computable* if it satisfies any of the following equivalent conditions:

1. There exists a computable binary function g with domain $\mathbb{N} \times \mathbb{N}$, such that for every $x \in \mathbb{N}$,
$$\lim_{s\to\infty} g(x,s) = \chi_S(x).$$
2. S is definable both by a Σ_2 formula and by a Π_2 formula. That is, there exist decidable subsets T and U of $\mathbb{N}^3$ for which, for all $x \in \mathbb{N}$,
$$x \in S \iff \exists y \forall z\, T(x,y,z) \iff \forall v \exists w\, U(x,v,w).$$
3. χ_S can be computed by a Turing machine which has access to an oracle for the halting problem $\emptyset'$.

Of course, (1) is the reason for the term "limit-computable." (Here the values of the computable function g are natural numbers, and so the equality means that $g(x, s) = \chi_S(x)$ for all but finitely many s.) In (2), the term Σ_2 refers to a formula of the first form, with quantifiers $\exists\forall$ in that order, followed by a decidable predicate; and Π_2 refers to the second type of formula, with quantifiers $\forall\exists$ before the decidable predicate. (The 2 denotes the number of quantifiers, and Σ and Π reflect whether the first one is $\exists$ or $\forall$. So Π_5 formulas are of the form $\forall y_1 \exists y_2 \forall y_3 \exists y_4 \forall y_5 R(x, y_1, \ldots, y_5)$, for example.)

In (3) we meet the important notion of *Turing reducibility*: a set A is *Turing-reducible to* B, written $A \leq_T B$, when there is a Turing machine which can compute χ_A under the assumption that it knows the answers to all questions of the form "is $n \in B$?" We usually think of such a program as having an *oracle* for B, which it consults to answer these questions, and we view such a program as a demonstration that A is no more difficult to decide than B, because any method of deciding B could be plugged into this oracle Turing machine and used to decide A as well. (So (3), expressed simply as "$S \leq_T \emptyset'$," says that S is no harder to decide than the halting problem.) When we mentioned earlier in this section that A and $\overline{A}$ are equally difficult to decide, we had Turing reducibility in mind. The reader should see quickly that χ_A can be computed with access to an $\overline{A}$-oracle and vice versa, so that $A \leq_T \overline{A}$ and $\overline{A} \leq_T A$. We call A and $\overline{A}$ *Turing-equivalent* for this reason, writing $A \equiv_T \overline{A}$, and we view A and $\overline{A}$ as having the same computational complexity.

The easiest way to see that a computable field F must have a limit-computable basis is to use condition (3). Given elements x_m and $x_{n_0}, \ldots, x_{n_k}$ of a computable field, one can readily write a program which searches for a polynomial showing x_m to be algebraic over $x_{n_0}, \ldots, x_{n_k}$. Indeed, a single program can do this uniformly in the inputs m and $n_0, \ldots, n_k$. Therefore, with access to a $\emptyset'$-oracle, one can ask whether that program would halt on those inputs, thus determining whether x_m is algebraic over $x_{n_0}, \ldots, x_{n_k}$ or not. So the $\emptyset'$-oracle allows us to decide membership in one particular transcendence basis B, where $x_n \in B$ iff x_n is transcendental over $x_0, \ldots, x_{n-1}$.

A similar argument shows that every computable group has a limit-computable maximal independent set. However, both for torsion-free groups and for fields of characteristic 0, we can actually do a bit better: from any limit-computable independent set B, we can get an independent set $\tilde{B}$ with the same span as B, such that $\tilde{B}$ is definable by a Π_1 formula. Recall, using the definition above, that a formula is Π_1 if it has a single universal quantifier.

In this case its negation is a Σ_1 formula, which must define a computably enumerable set, by Fact 3.2. Therefore, sets defined by Π_1 formulas are often called *co-c.e.*, being the complements of computably enumerable sets.

A co-c.e. set C is a set which begins life containing all natural numbers, and from which we then remove elements as they enter its complement. The actual elements of C are those which never get removed (that is, never enter the complement). Now let B be a limit-computable basis for a computable field F, so $\chi_B(x_n)$ is given as the limit of some computable function $g(n, s)$ as $s \to \infty$. If we find an $s \geq n$ for which $g(n, s) = 0$, then we remove x_n from the Π_1-basis $\tilde{B}$ we wish to build. If later on $g(x_n, s') = 1$ for some $s' > s$, of course we cannot restore x_n to $\tilde{B}$, since we wish $\tilde{B}$ to be co-c.e. However, we can enumerate the elements $1, 2, 3, \ldots$ of F until we find an m for which $x_n + m$ has not yet been removed from $\tilde{B}$. (After all, at most s'-many elements have been removed by stage s'.) Then we shift $x_n + m$ into the role previously held by x_n, and keep it in $\tilde{B}$ until a subsequent stage s'' (if any) at which $g(x_n, s'') = 0$ again. Since the function g only shifts back and forth between 0 and 1 finitely many times as $s \to \infty$, it is clear that if $x_n \in B$, then eventually some $x_n + m$ will be kept in $\tilde{B}$ forever, whereas if $x_n \notin B$, then eventually $g(x_n, s)$ stabilizes at 0 and no further elements $x_n + m$ are placed into the role of x_n. Thus, if $x_{k_0}, x_{k_1}, \ldots$ are the actual elements of B, then the elements of $\tilde{B}$ will be $x_{k_0} + m_0, x_{k_1} + m_1, \ldots$, for some elements $m_0, m_1, \ldots$ each of which is just a natural number in F. Clearly this $\tilde{B}$ is also algebraically independent and has the same span as B, hence is a transcendence basis for F, and by construction $\tilde{B}$ is co-c.e., hence Π_1-definable.

For computable groups, the exact same idea works, replacing each a_n with $(a_n)^m$ in the construction of the Π_1 maximal independent set. The only difference is in terminology: for fields, a (transcendence) basis is simply a maximal algebraically independent set, whereas for groups, the analogous object is just called a maximal independent set, with the term *basis* usually reserved for a maximal independent set which actually generates the entire group. If a group has a basis (under this definition), then the group is just the free group on the elements of that basis. Thus, not all groups have bases. The analogue for fields is sometimes called a *pure transcendence basis*: a transcendence basis (over some given subfield, often the prime subfield) which generates the entire field over that subfield. A field extension F which has a pure transcendence basis B over a subfield E must be a *purely transcendental extension* of E: that is, $F = E(B)$ is just the set of all E-rational functions, with elements of B as the variables. As with bases for groups, not all field extensions have pure transcendence bases.

This close analogy between groups and fields leads one to ask whether the same holds when we consider a basis for a group and a pure transcendence basis for a field (over its prime subfield). At this point, some disparity occurs between the two. Indeed, there is a disparity just within groups, if we compare free groups to free abelian groups. The following result should be challenging but not impossible, for a group theorist who has read this far.

Theorem 5.2 *Every computable free abelian group has a Π_1 basis.*

In contrast, results for computable free (non-abelian) groups have recently been derived by a large team of researchers at Notre Dame.

Proposition 5.3 (see [2]) *Every computable free group has a Π_2 basis.*

Theorem 5.4 (McCoy & Wallbaum, [10]) *There is a computable free group with no Σ_2 basis.*

Since every limit-computable set is both Π_2 and Σ_2, this shows that Theorem 5.2 fails when the free group is not abelian. Of course, such a group does still have a Π_1 maximal independent subset, by the results discussed above, but in the computable group built by McCoy and Wallbaum, no Π_1 maximal independent subset generates the entire group; indeed, no Σ_2 such set does. (Every Π_1 set is trivially Σ_2, just by adding a vacuous $\exists$ quantifier at the start of the defining formula.)

The analogous question for fields is the subject of current study: must a computable field F, purely transcendental over the rationals, have a Π_1 transcendence basis which generates all of F? One can pose the same question for purely transcendental field extensions more generally. For example, if F is purely transcendental over a computable subfield E, must F be generated over E by some Π_1 transcendence basis (for F over E)? And likewise, if the subfield E is only computably enumerable, not necessarily computable, must F be generated by some Π_1 (or even by some Π_2) transcendence basis over that E? The research group involved in this project does expect to have answers soon, but as of this writing, these questions remain open.

6 Galois groups

In this section we focus entirely on fields, asking to what extent we can use computability theory to examine the automorphism group of a field, or of a field extension over a fixed ground field. This subject was omitted from the talk which was the genesis of this paper, but was the subject of a question afterwards from Miasnikov, for which the author is grateful. Most of the questions considered here can also be asked about automorphisms of groups, but we leave that study for another time.

When a computable field F has infinite transcendence degree, its automorphisms can be extremely difficult to compute. Determining transcendence of a single element (over the prime subfield of F: either $\mathbb{Q}$ or $\mathbb{F}_p$, depending on characteristic) requires a $\emptyset'$-oracle, and we have discussed above some of the issues about finding a transcendence basis for F. Moreover, things get worse: a bijection from one transcendence basis to another need not extend to an automorphism of F, and determining whether or not it does so extend often requires much stronger oracles. (As a contrast, consider the case of computable vector spaces over $\mathbb{Q}$: for these structures, if any one basis B is computable, then each automorphism f is Turing-equivalent to the basis $f(B)$, and this gives a bijection between automorphisms and bases.) We will restrict ourselves to the far-more-accessible situation of automorphisms of a computable algebraic field F, by which we mean that F is a computable field which is algebraic over its prime subfield.

When we consider the automorphisms of F as a group, rather than individually, another difficulty arises: this group may be uncountable! The algebraic closure $\overline{\mathbb{Q}}$ of $\mathbb{Q}$, for example, is isomorphic to a computable field, yet its automorphism group is readily seen to have size continuum. So there is no hope for a theorem that the Galois group of a computable algebraic field must be isomorphic to a computable group. Moreover, when elements $f \in \text{Aut}(\overline{\mathbb{Q}})$ are viewed as bijections on $\mathbb{N}$ (with $x_n \mapsto x_{f(n)}$, on some computable presentation of $\overline{\mathbb{Q}}$), they can have arbitrarily large Turing degrees, so it is also not reasonable to hope to consider all automorphisms individually.

To consider automorphisms *en masse*, we must use a different method. We define the *full Galois action* A_F of F to be essentially the relation of being in the same orbit under automorphisms:

$$A_F = \{\langle\langle y_0, \ldots, y_n\rangle, \langle z_0, \ldots, z_n\rangle\rangle \in (F^*)^2 : (\exists \sigma \in \text{Aut}(F))(\forall i)\sigma(y_i) = z_i\}.$$

Recall that F^* denotes the set of all finite ordered tuples of elements of

F. Since F is algebraic, we can use an effective version of the Theorem of the Primitive Element to dispense with the tuples, by showing that A_F is Turing-equivalent to the simpler set

$$B_F = \{\langle a, b\rangle \in F^2 : (\exists \sigma \in \mathrm{Aut}(F))\sigma(a) = b\},$$

which we call the *Galois action* of F. These sets are discussed (and the results mentioned are proven) by Miller and Shlapentokh in [15], in relation to the question of computable categoricity for algebraic fields. They constitute a reasonable way to describe the Galois group, in the sense that if $\langle a, b\rangle \in A_F$, then with an A_F-oracle, one can build some $\sigma \in \mathrm{Aut}(F)$ with $\sigma(a) = b$. To do this, enumerate F as $\{a, x_0, x_1, x_2, \ldots\}$ and, given $\sigma \restriction \{a, x_0, \ldots, x_{n-1}\}$, search for an $x \in F$ with $\langle\langle a, x_0, \ldots, x_n\rangle, \langle b, \sigma(x_0), \ldots, \sigma(x_{n-1}), x\rangle\rangle \in A_F$. By induction (and the definition of A_F), such an x must exist, and when it appears, define $\sigma(x_n) = x$. So this method imparts some ability to discuss individual automorphisms as well: of course, not every single automorphism will be A_F-computable, but A_F gives enough power to extend arbitrary finite partial automorphisms to full automorphisms (or to determine that the finite partial automorphism cannot be so extended). From a group-theoretic point of view, one can see this as a process applicable to profinite groups in general, and it would likely be of interest to investigate Turing computability along these lines.

Now we consider the complexity of B_F (or equivalently A_F), for an arbitrary computable algebraic field F with prime subfield Q. It is well-known for such fields (and a proof is given in [15]) that a pair $\langle a, b\rangle$ lies in B_F iff a and b are conjugate over Q and, for every polynomial $p(X, Y) \in Q[X, Y]$, the polynomial $p(a, Y)$ has a root in F iff $p(b, Y)$ does. Writing this out, one sees that B_F is a Π_2 set (that is, definable by some Π_2 formula). We propose to show here, in moderate detail, that this is the best one can do, in terms of the complexity of formulas: B_F need not be Σ_2.

Σ_2 sets are normally seen as being defined by a property which says that something is finite. Although the actual definition of a Σ_2 formula says nothing of the sort, one can show that for every set S defined by any Σ_2 formula, there exists a computable function f with domain $\mathbb{N}$ for which

$$(\forall x \in \mathbb{N})[x \in S \iff W_{f(x)} \text{ is finite}].$$

Conversely, every set S which has such a function f is Σ_2, since the finiteness of $W_{f(x)}$ is expressible as a Σ_2 formula about x. For a proof, see [19, Theorem IV.3.2]. So we may treat the statement "$x \in S$" as equivalent to the statement that when we enumerate the c.e. set $W_{f(x)}$, only finitely many elements will ever appear in this set.

Therefore, in order to build a computable algebraic field F with B_F not Σ_2, we need only ensure that for every partial computable function φ_e, there exists a pair $\langle a, b \rangle$ which lies in B_F iff $W_{\varphi_e(\langle a,b \rangle)}$ is infinite. This will show that φ_e cannot serve as the function f which reduces B_F to the finiteness condition given above. If we can do this for every partial computable function φ_e, then there will be no computable function which can play the role of f, and thus B_F will not be Σ_2. (Of course, we really only need to do so for every computable function whose domain is all of $\mathbb{N}$, but it is not possible to distinguish these from the other partial computable functions effectively. So we simply do it for every φ_e.)

In fact, a field F with all these properties is already available to us. In [13, Theorem 4.1], the author constructed two isomorphic computable algebraic fields F and $\tilde{F}$ which had the property that no limit-computable function could be an isomorphism from F onto $\tilde{F}$. (This yielded a result about $\emptyset'$-computable categoricity, giving a further example of the relation between categoricity issues and complexity of the Galois action.) In that field F, for every e, the e-th rational prime p_e has two square roots, denoted by $\pm\sqrt{p_e}$. In order to ensure that $\lim_s \varphi_e(x, s)$ is not an isomorphism from F onto $\tilde{F}$, the construction made sure that if $\varphi_e(\sqrt{p_e}, s)$ was one of the two square roots of the e-th prime $\tilde{p}_e$ in $\tilde{F}$, then some polynomial $q(X, Y) \in \mathbb{Q}[X, Y]$ existed for which $q(\sqrt{p_e}, Y)$ had no root in F, yet $q(\varphi_e(\sqrt{p_e}, s), Y)$ had a root in $\tilde{F}$. (To keep F and $\tilde{F}$ isomorphic, $q(-\sqrt{p_e}, Y)$ had a root in F, and $q(-\varphi_e(\sqrt{p_e}, s), Y)$ had no root in $\tilde{F}$. So an isomorphism existed, but it mapped $\sqrt{p_e}$ to $-\varphi_e(\sqrt{p_e}, s)$, not to $\varphi_e(\sqrt{p_e}, s)$.)

If, for some $t > s$, $\varphi_e(\sqrt{p_e}, t)$ turned out to equal the other square root of $\tilde{p}_e$ in $\tilde{F}$, then roots of $q(\sqrt{p_e}, Y)$ and $q(-\varphi_e(\sqrt{p_e}, s), Y)$ were added to F and $\tilde{F}$, and a new polynomial was selected in place of q to make $\varphi_e(\sqrt{p_e}, t)$ the wrong choice. So, if $\lim_s \varphi_e(\sqrt{p_e}, s)$ actually did converge to a limit, then that limit was the wrong square root of $\tilde{p}_e$ in $\tilde{F}$, and thus the limit-computable function $\lim_s \varphi_e(x, s)$ could not be an isomorphism. (This was done simultaneously for every e, using a different prime p_e for each.)

For our purposes here, the useful part of this construction is that the pair $\langle \sqrt{p_e}, -\sqrt{p_e} \rangle$ turned out to lie in B_F iff, as $s \to \infty$, the value of $\varphi_e(\sqrt{p_e}, s)$ went back and forth infinitely often between the two square roots of $\tilde{p}_e$ in $\tilde{F}$.

If the limit stabilized, or if $\varphi_e(\sqrt{p_e}, s)$ was simply undefined for some s, then the construction made no more changes to F and $\tilde{F}$ after the stabilization occurred, or after the stage s for which $\varphi_e(\sqrt{p_e}, s)$ was undefined. With no more changes made, there remained a polynomial q for which $q(\sqrt{p_e}, Y)$ had a root in F iff $q(-\sqrt{p_e}, Y)$ did not.

So we see that for every $e \in \mathbb{N}$, $\langle \sqrt{p_e}, -\sqrt{p_e} \rangle \in B_F$ iff there exist infinitely many stages s in the construction of F at which:

- $\varphi_e(\sqrt{p_e}, s) = \sqrt{\tilde{p}_e}$, and
- $\varphi_e(\sqrt{p_e}, s+1) = -\sqrt{\tilde{p}_e}$.

That is to say, $\lim_s \varphi_e(\sqrt{p_e}, s)$ did indeed switch back and forth between $\pm\sqrt{\tilde{p}_e}$ infinitely often. Thus, in this F, we see that for each $e \in \mathbb{N}$, $\langle \sqrt{p_e}, -\sqrt{p_e} \rangle \in B_F$ iff something happens *infinitely* often, suggesting that B_F cannot be defined by a Σ_2 formula saying that something happens only finitely often. We do not propose to give here all of the details necessary for understanding the rest of this argument; they can be found in [19, Chapter IV]. Briefly, the set T of those e for which this switching happened infinitely often is readily shown to be a Π_2-complete set. But then B_F is Π_2-complete, hence cannot be Σ_2. So we have completed a proof of the following.

Theorem 6.1 *There exists a computable algebraic field F whose Galois action B_F is not Σ_2 (and therefore not limit-computable). Indeed, B_F is Π_2-complete.* ■

We mention here, without giving a proof, that it is not difficult to adapt the proof of [13, Theorem 4.1] to yield a single computable algebraic field F such that $\mathrm{Aut}(F)$ has continuum many automorphisms, among which none except the identity is limit-computable. That proof can readily be adjusted to have $\tilde{F}$ be the same field as F, and the requirement that $\lim_s \varphi_e(x, s)$ not be an automorphism is broken into infinitely many requirements of the form:

$$\mathcal{R}_{e,j}: \quad \text{If } (\exists y < j)[\lim_s \varphi_e(y, s) \neq y], \text{ then } \lim_s \varphi_e(x, s) \text{ is not an isomorphism.}$$

So one simply follows the strategy of [13, Theorem 4.1] for $\mathcal{R}_{e,j}$ whenever it appears that a j exists for which the limit-computable function in question is *not* the identity on $\{0, \ldots, j-1\}$. The rest of the details are left to the reader.

We promised above that, although these results show that nontrivial automorphisms of a computable algebraic field F need not be limit-computable, there must be some which are close to being so. To make this precise, we relativize the construction of the halting problem. Recall that in Section 3, we built the set $\emptyset'$, the halting problem, by asking whether a given partial computable function ever halts on a given input. Now we repeat the same process relative to an oracle set A, asking whether a given partial A-computable function Φ_e^A ever halts on the input e. This problem, denoted by A', is Turing-equivalent to the halting problem relative to A:

$$A' = \{e \in \mathbb{N} : \Phi_e^A(e) \text{ halts}\} \equiv_T \{\langle e, x\rangle \in \mathbb{N}^2 : \Phi_e^A(x) \text{ halts}\}.$$

This explains the notation $\emptyset'$ used above: the halting problem relative to a computable oracle, such as $\emptyset$, is just the original halting problem. It is readily seen that, for all sets $A \subseteq \mathbb{N}$, we have $A' \not\leq_T A$, by the exact same argument as before. (On the other hand, $A \leq_T A'$.) The set A' is known as the *jump* of A, usually pronounced "A-jump." The jump of a function is defined to be the jump of its graph, viewed as a subset of $\mathbb{N}$ under the usual pairing function. The jump operation respects Turing equivalence, and so it is also reasonable to speak of the *jump* $\boldsymbol{d}'$ *of a Turing degree* $\boldsymbol{d}$.

However, the jump operation fails to respect Turing nonequivalence. It has long been known that there are noncomputable sets A for which $A' \equiv_T \emptyset'$. Such sets are said to be *low*, since they are very close to being computable: they have the same jump (up to Turing equivalence) as a computable set. Likewise, there exist sets $B <_T \emptyset'$ with $B' \equiv_T \emptyset''$, the second iterate of the jump on $\emptyset$; such a set B is called *high*, and is thought of as being very close to $\emptyset'$ in Turing computability, since the two have the same jump. (One would also then say that $\emptyset'$ is *low relative to* B, since $(\emptyset')' \leq_T B'$.)

The famous Low Basis Theorem of Jockusch and Soare then gives the result we promised: that if a computable algebraic field F has nontrivial automorphisms, then it has an automorphism of degree "not much more than" $\emptyset'$. In particular, it has an automorphism f with jump $f' \leq_T \emptyset''$, so that f itself is low relative to $\emptyset'$. We sketch a quick proof. Consider the automorphism tree I_{FF} for F, as defined in [13, Section 5]. Since F has a nontrivial automorphism g, fix the least y with $g(y) \neq y$, and define the computable subtree T containing all nodes $\sigma \in I_{FF}$ with $\sigma(y) \neq y$. The path g through T shows that this subtree is infinite. Clearly T is still finite-branching, and $\emptyset'$ can compute the size of each level of T, so by the Low

Basis Theorem relativized to $\emptyset'$, there is a path f through T which is low relative to $\emptyset'$. For the original proof of the Low Basis Theorem, see [7].

It is natural to ask whether similar results apply to automorphisms of groups: if G is a computable group, perhaps with certain specific properties, then what can be said about the automorphisms of G? Most likely, investigations of this question would diverge from the results given in this section. We have focused here on algebraic fields, and have used heavily the facts that in such fields we can identify each element very accurately by finding its minimal polynomial $p(X)$ over the prime subfield. This accuracy is not absolutely exact, since $p(X)$ may have other roots in the field. However, there is a finite bound (just the degree of $p(X)$) for the number of such roots, and the existence of this bound facilitates the use of the Low Basis Theorem. It is not clear whether there is any property for groups, analogous to algebraicity, which would allow us to apply the Low Basis Theorem to the class of computable groups with that property. The most obvious analogue of an algebraic field is a torsion group, but such groups do not have the relevant property of algebraic fields: under automorphisms of an algebraic field, each orbit is finite, whereas for torsion groups infinite orbits are possible. Of course, it would still be natural to begin these investigations with better-known classes of groups: torsion groups and/or abelian groups, for example, perhaps of finite rank. We leave such questions for future study, and we encourage group theorists to go through the literature on computable groups, to consider these issues, and to offer new questions of interest from the standpoint of group theory.

References

[1] W.W. Boone; The word problem, *Annals of Mathematics* **70** (1959), 207–265.

[2] J. Carson, V. Harizanov, J. Knight, K. Lange, C. Maher, C. McCoy, A. Morozov, S. Quinn, & J. Wallbaum; Describing free groups, to appear in the *Transactions of the American Mathematical Society.*

[3] Yu.L. Ershov; Theorie der Numerierungen, *Zeits. Math. Logik Grund. Math.* **23** (1977), 289–371.

[4] M.D. Fried & M. Jarden, *Field Arithmetic* (Berlin: Springer-Verlag, 1986).

[5] A. Frohlich & J.C. Shepherdson; Effective procedures in field theory, *Phil. Trans. Royal Soc. London, Series A* **248** (1956) 950, 407-432.

[6] W. Hodges; *A Shorter Model Theory* (Cambridge: Cambridge University Press, 1997).

[7] C.G. Jockusch & R.I. Soare; Π^0_1-classes and degrees of theories, *Transactions of the American Mathematical Society* **173** (1972), 33-56.

[8] L. Kronecker; Grundzüge einer arithmetischen Theorie der algebraischen Größen, *J. f. Math.* **92** (1882), 1-122.

[9] M. Lerman, *Degrees of Unsolvability: Local and Global Theory* (Berlin: Springer-Verlag, 1983).

[10] C. McCoy & J. Wallbaum; Describing free groups, part II: Π^0_4 hardness and no Σ^0_2 basis, to appear in the *Transactions of the American Mathematical Society.*

[11] G. Metakides & A. Nerode; Effective content of field theory, *Annals of Mathematical Logic* **17** (1979), 289-320.

[12] R. Miller; Computable fields and Galois theory, *Notices of the American Mathematical Society* **55** (August 2008) 7, 798-807.

[13] R. Miller; $\boldsymbol{d}$-Computable categoricity for algebraic fields, *The Journal of Symbolic Logic* **74** 4 (2009), 1325–1351.

[14] R. Miller; Computability and differential fields: a tutorial, in *Differential Algebra and Related Topics: Proceedings of the Second International Workshop*, eds. L. Guo & W. Sit (World Scientific, 2011), ISBN 978-981-283-371-6.

[15] R. Miller & A. Shlapentokh; Computable categoricity for algebraic fields with splitting algorithms, submitted for publication.

[16] P.S. Novikov; On the algorithmic unsolvability of the word problem in group theory, *Trudy Mat. Inst. Steklov* **44** (1955).

[17] M. Rabin; Computable algebra, general theory, and theory of computable fields, *Transactions of the American Mathematical Society* **95** (1960), 341-360.

[18] H. Rogers, Jr.; *Theory of Recursive Functions and Effective Computability* (New York: McGraw-Hill Book Co., 1967).

[19] R.I. Soare; *Recursively Enumerable Sets and Degrees* (New York: Springer-Verlag, 1987).

[20] V. Stoltenberg-Hansen & J.V. Tucker, Computable Rings and Fields, in *Handbook of Computability Theory*, ed. E.R. Griffor (Amsterdam: Elsevier, 1999), 363-447.

[21] B.L. van der Waerden; *Algebra*, volume I, trans. F. Blum & J.R. Schulenberger (New York: Springer-Verlag, 1970 hardcover, 2003 softcover).

Department of Mathematics
Queens College – C.U.N.Y.
65-30 Kissena Blvd.
Flushing, New York 11367 U.S.A.
Ph.D. Programs in Mathematics & Computer Science
The Graduate Center of C.U.N.Y.
365 Fifth Avenue
New York, New York 10016 U.S.A.

E-mail: `Russell.Miller@qc.cuny.edu`
Website: `qcpages.qc.cuny.edu/~rmiller`