TURING MACHINES, CAYLEY GRAPHS, and INESCAPABLE GROUPS

by

Aubrey da Cunha

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mathematics)
in The University of Michigan
2012

Doctoral Committee:

Professor Andreas Blass, Chair
Professor John Stembridge
Associate Professor Kevin Compton
Associate Professor Yaoyun Shi
Assistant Professor Scott Schneider

© Aubrey da Cunha 2012
All Rights Reserved

TABLE OF CONTENTS

LIST OF FIGURES . . . . . **iv**

LIST OF TABLES . . . . . **v**

CHAPTER

I. Introduction 1

1.1 Notation and Terminology 4

II. Turing Machines on Tape Graphs 5

2.1 General Tape Geometries 5
2.1.1 The word problem for tape graphs 8
2.2 Turing Machines on Cayley Graphs 9
2.2.1 A Well-ordering in Trees 10
2.2.2 Tree of Super-reduced Words 12

III. Power of Turing Machines on Tape Graphs 15

3.1 Simulation of Standard Turing machines 15
3.2 Proof of Theorem III.1 20
3.2.1 Computation Step 23
3.2.2 Leftward Step 23
3.2.3 Rightward Step 26
3.3 Keeping Track of Memory 33
3.4 An Alternative Characterization of the Turing Degrees 39
3.4.1 Further Remarks 40

IV. Efficiency of Turing Machines on Tape Graphs 42

4.1 Computational Complexity over Cayley Graphs 42
4.2 Efficiency of Theorem III.1 43
4.3 Proof of Theorem IV.1 44
4.3.1 Machines over G can simulate machines with oracles 44
4.3.2 Machines with oracles can simulate machines over G 48
4.4 Connections with Combinatorial Group Theory 49
4.4.1 A Polynomial-time analog of the Boone-Novikov Theorem 50
4.4.2 Nearly Linear Time 51
4.4.3 Groups of Polynomial Growth 53

V. Inescapable Groups 55

5.1 What is an Inescapable Group? 55

5.2 Construction of an Inescapable Group 55
5.2.1 A Combinatorial Lemma 57
5.2.2 The Golod-Shafarevich Theorem 59
5.2.3 The Construction 60
5.3 Inescapable Groups are Burnside Groups 64

BIBLIOGRAPHY **69**

LIST OF FIGURES

Figure

2.1 Example of a tape graph 7
3.1 Portion of the tree during a leftward step 24
3.2 Portion of the tree during a rightward step when the tape head is not at h and F is empty 27
3.3 Portion of the tree during a rightward step when the tape head is not at h and F is not empty 28
3.4 Portion of the tree during a rightward step when the tape head is at h and A_h has no elements greater than x 30
3.5 Portion of the tree during a rightward step when the tape head is at h and A_h has some element greater than x 32

LIST OF TABLES

Table

3.1 Transition Table for N 19
3.2 Transition Table for N 36

CHAPTER I

Introduction

When Alan Turing originally defined his a-machines, which would later be called Turing machines [20], he envisioned a machine whose memory was laid out along a one-dimensional tape, inspired by the ticker tapes of the day. This seemed somewhat arbitrary and perhaps unduly restrictive, and so, very quickly, machines with multiple and multi-dimensional tapes were proposed. The focus at the time was defining the term "computable", and as adding tapes and dimensions defined the same class of functions as Turing's original, simpler model, studying alternate tape geometries fell out of favor for some time.

The complexity theory community then reignited interest in alternative tape geometries by considering not the class of functions computable by Turing machines, but time and space complexity of functions on different tape geometries. This led to a number of results about relative efficiency of machines with one/many tapes and one/two/high-dimensional tapes. For example, the language of palindromes (those strings that read the same forward and backward) can be computed in $O(n)$ time on a two-tape machine, but requires $\Omega(n^2)$ on a one-tape machine with one read/write head. Or, an m-dimensional Turing machine running in time $T(n)$ can be simulated by a k-dimensional Turing machine ($k < m$) in time $T(n)^{1+\frac{1}{m}-\frac{1}{k}+\epsilon}$ for all $\epsilon > 0$

[8]. Related to this is the belief that quasi-linear time is not robust for different dimensions of tape [13].

Many of the proofs and algorithms used in the study of multidimensional Turing machines make their way into or are inspired by the world of mesh-connected systems. Mesh-connected systems are arrays of identical, relatively dumb processors (typically with memory logarithmic in the input size) that communicate with their neighbors to perform a computation. Time use on a Turing machine with a d-dimensional tape is intimately tied to the power use of a d-dimensional mesh of processors with finite memory, so there is some natural crossover. Mesh-connected systems constitute an area of very active research now, but since it remains very closely tied to the physical implementation, research is generally restricted to two- and three-dimensional grids.

It is natural, then, to ask how far we can push the notion of a tape. In this thesis, we go beyond the world of rectangular grids and consider tapes at their most general. The purpose of this is two-fold. First, to give the complexity theorists a general framework in which to work, subsuming all current tape-based Turing models. Second, to provide some evidence that alternative tape geometries are interesting from a recursion-theoretic perspective.

The tapes we will consider will follow very closely in the original tradition of Turing. This is in contrast to the approaches of Schönhage [18] and Kolmogorov and Uspenskii [10]. In their pointer machine models, the tape is a dynamic object and computation is not done *on* the tape, but *with* the tape. Instead of writing on the cells of the tape, information is encoded directly into the connections between the cells. This ability to rewire their tapes makes both of their models quite powerful, but also quite difficult to analyze.

Our tapes are static objects and our machines may merely write symbols on the

cells of the tape. Any tape of this type can be modeled as a digraph with nodes corresponding to tape cells and edges corresponding to allowable transitions. Hence, we start by introducing a number of graph-theoretic conditions that ought to be satisfied by a reasonable tape. It turns out that the criteria we outline are necessary and sufficient conditions for the graph to be the Cayley graph of a finitely generated, infinite group.

We then turn to the question of whether allowing arbitrary Cayley graphs as tapes is just another equivalent machine model for the class of computable functions. Interestingly, this will depend on the structure of the group from which the Cayley graph is generated. For groups with solvable word problem, this does indeed lead to machines that compute the class of computable functions, however for groups with unsolvable word problems, these machines are strictly more powerful than standard Turing machines. In fact, they can be as powerful as any oracle machine and we end up with an alternative definition of the Turing degrees that is machine based and doesn't rely on oracles.

We observe at this point that the simulations used to show the power of machines over arbitrary graphs are, in fact, efficient. This allows us to draw connections between resource bounded computations over Cayley graphs and well-known time and space classes. A side effect of this is the observation that attempts at using very low-level combinatorial arguments to defeat relativization barriers will have to make essential use of the structure of the underlying tape.

The constructions and proofs of these results begin to raise questions about what kind of computable objects we can hope to find in arbitrary Cayley graphs. In particular, whether we can always find an infinite, computable, non-self-intersecting path. We call such a path an escape and we construct a group without any escapes.

This construction is non-trivial as any group without an escape must consist entirely of elements of finite order, yet be infinite. Such a group is called a Burnside group and the existence of Burnside groups was an open problem for some time.

1.1 Notation and Terminology

The Cayley graph of a group, G, relative to a generating set, $S \subset G$, is the graph (V, E) with $V = G$ and $E = \{(g, gs)\}_{g \in G, s \in S}$. We can think of this as a colored, directed graph where the edge (g, gs) has color s.

Our oracle machines will have a separate, write-only query tape and a special query state. When the finite control transitions to the query state, it will then immediately transition either to the "yes" or "no" state according to whether the string written on the query tape belongs to the oracle or not. The query tape is then automatically erased. For our purposes, this is *not* equivalent to the formalism in which the oracle is written on a separate read-only tape.

Trees will always grow upwards with the root at the bottom and the leaves at the top. So, terms like "above" (resp. "below") mean further from (resp. closer to) the root. Ancestors of a node are those between it and the root while descendants are those nodes for which it is an ancestor.

Finally, we will denote disjoint unions by $\sqcup$ to distinguish them from regular unions, $\cup$. We will always denote the identity element of a group by λ. We will also denote the empty string by λ. This will not cause confusion as the identity element of a group of strings is the empty string. All of our logarithms will be binary logs.

CHAPTER II

Turing Machines on Tape Graphs

2.1 General Tape Geometries

Any tape essentially consists of a collection of cells, each of which can hold a single symbol, together with a mechanism for moving from one cell to another. So underlying any tape is an edge-colored digraph. Edges in this graph represent allowable transitions between cells and the edge coloring encodes directionality. In conjunction with a finite control, this directionality is equivalent to the conditions on the stored symbol and control state under which that particular transition between cells occurs. In order to be a reasonable Turing tape, this digraph should satisfy a few restrictions.

1. Uniqueness of outgoing colors - From any vertex, there should be exactly one outgoing edge of each color. Since the mechanism by which the tape head moves is encoded in the edge color, outgoing edges should have different colors. Also, since the transition function is independent of the tape cell, the collection of colors going out from each vertex should be the same.

2. Infinity - The tape should have an infinite number of cells. Otherwise, the machine is just a finite automaton.

3. Connectedness - As the head moves during the course of a computation, it remains on a single connected component. Inaccessible cells are useless, so

we can require that every tape cell be accessible from the starting point. In particular, this means that the graph is connected.

4. Finitely many colors - The transition function of a TM is finite, so there should only be finitely many outgoing colors. Having more colors doesn't change the computational power, since only finitely many of them could be referenced by the transition table anyway.

5. Backtracking - The Turing machine should be able to return to the cell it just came from. More than that, the machine should know *how* to return to the cell it just came from. The most direct way to do this is to insist that for each color, c, there is some color c^{-1} such that whenever there is an edge of color c from v to w, there is an edge of color c^{-1} from w to v and vice-versa.

Let us then make this into a definition.

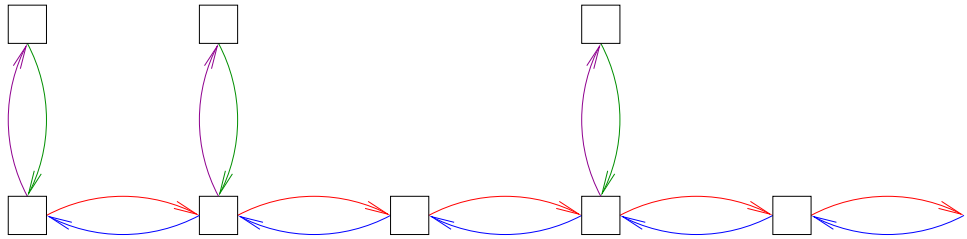
Definition II.1. Let G be an edge-colored digraph satisfying 1–5 and let S be the set of edge colors. Then G is a *tape graph* and is denoted (G, S).

This is a hugely broad class of graphs. Perhaps even too broad. Consider, for example, tapes such as the one in figure 2.1. This looks like a standard one-dimensional tape, but with branches from some of the cells. We could encode any language, such as the halting problem, into the branches and since a Turing machine can easily detect whether a cell has a branch, we could make anything computable. This is kind of cheating.

So, we suggest another restriction that we would like tape graphs to satisfy.

6. Homogeneity - Every vertex should "look like" every other vertex. Technically, the subgroup of the automorphism group of the graph that preserves edge colors

Figure 2.1: Example of a tape graph



Edges not shown are assumed to be loops

should act vertex transitively. This is an extension of the assumption that all tape cells are indistinguishable, in this case by the local geometry.

This clearly rules out tapes such as the one in figure 2.1. In fact, it does more than this and the class of tape graphs satisfying 1–6 admits a much nicer characterization.

Theorem II.2. *Any tape graph satisfying 1–6 is the Cayley graph of an infinite, finitely-generated group relative to a symmetric generating set.*

The proof of this is a graph-theoretic exercise, so we won't include it here. We will simply point out that restrictions 1 and 6 imply that our tape is a Cayley graph, restrictions 2, 3, and 4 make it the Cayley graph of a finitely generated infinite group, and restriction 5 forces the generating set to be closed under inverses. In addition, any Cayley graph of a finitely generated infinite group with a generating set that is closed under inverses satisfies 1–6. This suggests that Cayley graphs are, in some sense, the "right" degree of generality and leads the definition,

Definition II.3. Let G be an infinite group and $S \subset G$ be a finite generating set for G that is closed under inverses. Then the Cayley graph of G generated by S is called the *Cayley tape*, (G, S).

Using this general type of tape, we can then ask questions about the structure of Turing machines with tapes given by assorted graphs. All of our results hold in

the more general setting of arbitrary tape graphs, but the real model of interest is that with Cayley tapes. Results about the limitations of Turing machines over tape graphs will be proved for the more general setting, while results on the existence of a tape graph with some property will generally be proved with the added property that the graph is a Cayley graph.

2.1.1 The word problem for tape graphs

Later, we will need to make reference to the edge relation for a tape graph. In particular, the difficulty in computing it. However, not all directed graphs come with a canonical encoding from vertices to natural numbers, so the phrase "computability of the edge relation" has no meaning. Instead, we will define a "word problem" over the tape graph, (G, S) with distinguished vertex v.

Let S^* be the free semigroup over S. Since S is symmetric, we get a group by adding the relations $ss^{-1} = e$ for each $s \in S$. We will call this group H. S^* acts on the right on G in a natural way: for any $w \in G$ and $s \in S$, ws is the vertex reached by following the edge colored s from w. This then extends to an action by S^* and since any edge of color s^{-1} is the reverse edge to an edge of color s, we get a compatible action of H on G. We will make extensive use of this action later on. Note that when G is a Cayley graph, this action is just multiplication on the right in the group.

Definition II.4. The *word problem* for (G, S) with distinguished vertex, v, is the problem of deciding, given $w, w' \in S^*$, whether $vw = vw'$.

When (G, S) is a Cayley tape, this coincides with the usual definition of the word problem for G. In fact, when (G, S) is a Cayley tape, the word problem is independent of v, so we may as well take v to be the identity in the group.

In the next section, when we define Turing machines over tape graphs, it will be natural for the distinguished vertex to be the starting location of the tape head. Thus, we will often omit either the starting location of the tape head or the distinguished vertex with the understanding that they are identical.

2.2 Turing Machines on Cayley Graphs

Definition II.5. Let (G, S) be a tape graph with $S = \{g_1, \ldots, g_n\}$. Then a Turing Machine over (G, S) is a 7-tuple, $(Q, \Gamma, b, \Sigma, \delta, q_0, F)$ where

- Q is the finite set of states
- Γ is the finite set of tape symbols
- $b \in \Gamma$ is a designated blank symbol
- $\Sigma \subset \Gamma \backslash \{b\}$ is the set of input symbols
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{g_1, \ldots, g_n\}$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subset Q$ is the set of terminal states (typically one to accept and one to reject)

This definition varies from the standard definition only in the interpretation of the transition function. Whereas a standard TM has a two-way infinite one-dimensional tape and the transition function includes instructions for moving left or right, a TM over (G, S) has as a tape the graph G and the transition function has instructions for moving along edges colored by an element of S. For example, a Turing Machine over $(\mathbb{Z}, \{-1, +1\})$ is a standard one-dimensional TM and a TM over $(\mathbb{Z}^2, \{(0, -1), (-1, 0), (0, 1), (1, 0)\})$ is a standard two-dimensional TM.

We have intentionally skipped the notion of how to provide input for machines of this type. Most generally, we could insist that the initial set of non-blank tape cells

be connected and contain the initial location of the tape head. However, we really intend these machines to work like Turing machines, and therefore, to compute on strings of symbols. It turns out that there will be a canonical way to lay out strings on the tape graph, but we need some machinery first.

2.2.1 A Well-ordering in Trees

We shall turn aside from the main topic for a moment to discuss a general statement about trees. There are many ways to define an order on the vertices of a tree, but we are going to be interested in the lexicographic order. In general, lexicographic orderings on trees have few nice properties, but we show that finitely branching trees have subtrees where the lexicographic order is in fact, a well-order.

First, some definitions. Let T be a finitely branching rooted tree. Denote by $[T]$ the set of all infinite directed paths through T and by $\sqsubset$ the partial ordering on vertices induced by the tree. Our convention will be that $v \sqsubset w$ means that v is closer to the root than w.

In order for the lexicographic ordering on T to even make sense, we must have a linear order on the set of children of each node. Denote the order on the children of $v \in T$ by $<_v$. Then the lexicographic order, $<$ is defined as follows:

- If $v \sqsubset w$, then $v < w$.
- If neither $v \sqsubset w$ nor $w \sqsubset v$, let u be the greatest lower bound of v and w according to $\sqsubset$ and let $u' \sqsubset v$ and $u'' \sqsubset w$ be children of u. Then $v < w$ if and only if $u' <_u u''$.

This order can, in fact, be extended to an order on $T \cup [T]$. Identifying elements of $[T]$ with subsets of T and elements of T with one-element subsets of T, we can

define

$$x < y \iff (\exists w \in y)(\forall v \in x) v < w$$

Defined in this way, $<$ is a linear order, but we can't really hope for any more structure than that. However, as promised, with a bit of pruning, we can find a subtree with much more structure.

Theorem II.6. *Let T be a finitely branching tree and let $<$ be the lexicographic ordering on the nodes of and paths through T as given above. Define*

$$T' = \{v \in T | \forall w \in [T], v < w\}$$

Then $<$ restricted to T' is order isomorphic to an initial segment of ω. In addition, if T is infinite, T' is infinite as well.

Proof. This follows from the following direct consequence of König's Lemma.

Lemma II.7. *Every element of T' has only finitely many $<$-predecessors.*

Proof. Suppose $v \in T'$ had infinitely many $<$-predecessors. Then we could form the tree,

$$S = \{w \in T' | w < v\}$$

This is, in fact, a tree since T' is a tree and $x \sqsubset y$ implies $x < y$. Since v has infinitely many $<$-predecessors, S is infinite.

By König's Lemma, there must be a path through S, call it P. But T' is a subtree of T so $P \in [T]$. By definition of $<$, $P < v$, but $v \in T'$ so $v < P$ by the definition of T. This is a contradiction, so v must have only finitely many predecessors. $\square$

Any linear order in which every element has only finitely many predecessors clearly cannot have an infinite descending chain, so must be a well-order. As $\omega + 1$ has an

element with infinitely many predecessors, the order type must be an initial segment of ω.

For the second part of Theorem II.6, we need an additional lemma.

Lemma II.8. *If $[T]$ is non-empty, then $[T]$ has a minimal element in the $<$ ordering.*

Proof. We can inductively construct the minimal element of $[T]$. For any $v \in T$, define $s(v)$ to be the minimal (according to $<_v$) child of v that is a member of some element of $[T]$ if such a vertex exists. Note that if there is a path through v, $s(v)$ is defined and there is a path through $s(v)$. If v_0 is the root, then $P = \{s^{(n)}(v_0)\}_{n\in\mathbb{N}}$ is the desired minimal element.

Since $[T]$ is non-empty, there is a path through the root and so, by induction, $s^{(n)}(v_0)$ is defined for all n. Therefore P is indeed a path.

To see that P is minimal, let $P \neq P' \in [T]$. Let $v = s^{(m)}(v_0)$ be the largest element (according to $\sqsubset$) of $P \cap P'$ and let $w \in P'$ be a child of v. By maximality of v, $w \neq s(v)$ and by construction, $s(v) \leq_v w$. Therefore, $s(v) <_v w$. By the lexicographic ordering, w is greater than all descendants of $s(v)$ and also greater than all ancestors of $s(v)$ (since ancestors of $s(v)$ are also ancestors of w). Therefore, $w > u$ for all $u \in P$ and $P' > P$. $\square$

Now, let T be infinite. Then, by König's Lemma again, $[T]$ is non-empty. Let P be the minimal path in $[T]$ according to Lemma II.8. Then $P \subset T'$ since for any $v \in P$ and $P' \in [T]$, $v < P \leq P'$. P is infinite, so T' is infinite as well. $\square$

2.2.2 Tree of Super-reduced Words

Recall that we are attempting to develop a canonical way of providing a Turing Machine on a potentially exotic tape with a string as input. Given a tree, the results of the previous section will provide us with a subtree in which the lexicographic

ordering is a well-ordering, so in this section we will identify a useful tree within any given tape graph.

We will routinely use the natural correspondence between sequences of colors and vertices introduced in section 2.1.1. Henceforth, sequences and vertices will be used interchangeably. Of course, multiple sequences will correspond to the same vertex, but we will only use the correspondence in the other direction.

Definition II.9. Let (G, S) be a tape graph. A *super-reduced word* is a finite sequence of elements of S such that no pair of initial segments correspond to the same vertex of G.

Super-reduced words correspond exactly to non-self-intersecting finite paths through the tape graph. If (G, S) is a Cayley tape, then being a super-reduced word is equivalent to having no subword whose product is the identity in G. Note that all prefixes of a super-reduced word are themselves super-reduced.

Form the tree, T, of super-reduced words with symbols from S. Since every vertex has at least one super-reduced word corresponding to it, the natural map from T to G is a surjection. Thus, since G is infinite, T is infinite as well. Therefore, we can construct an infinite T' as in Section 2.2.1 where the lexicographic ordering is a well-ordering. We are interested not in T', but in a subtree, which we will call R. To define R we will want another definition.

Definition II.10. We say that two sequences of colors, v and w, are equivalent with respect to G (and some distinguished vertex), or $v \equiv_G w$ if they correspond to the same vertex.

Now we can define R as follows,

$$R = \{v \in T' | (\forall w \in T')\, v \equiv_G w \implies v \leq w\} \tag{2.1}$$

That is, R is the set of words in T' that are lexicographically minimal among sequences that represent the same vertex.

It's not obvious at first glance, but R is a tree. Suppose to the contrary that $uv = w \in R$ but $u \notin R$. Then there is some $u' < u$ in R corresponding to the same vertex. Since we began with the tree of super-reduced words, u and u' are incomparable in the tree order. Therefore, u and u' must differ at some first location. So, $u'v \equiv_G w$ but uv and $u'v$ differ for the first time at the same location as u and u' and since $u' < u$, $u'v < uv = w$. This is a contradiction since w was supposed to be minimal among words corresponding to the same vertex, so R is indeed a tree.

Also non-obvious is the fact that R is infinite. By the proof of Theorem II.6, T' contains a path. As was noted earlier, since we began with the tree of super-reduced words, elements that are comparable in the tree order cannot correspond to the same vertex. Therefore, the path in T' corresponds to an infinite collection of vertices. R represents the same vertices since we only pruned redundant representations, so R also represents an infinite number of vertices and is therefore infinite.

Now, for any tape graph, we have a tree that is well-ordered lexicographically, represents infinitely many vertices and represents each individual vertex at most once. This is how we will provide input. Simply initialize the machine with the nth vertex in the well-order containing the nth symbol of the input.

CHAPTER III

Power of Turing Machines on Tape Graphs

One of the first questions to be asked about any new model of computation is whether the class of functions computable by the new model is different from the class of computable functions. For Turing machines on tape graphs, this depends rather sensitively on properties of the graph. For example, as we will see later, the word problem for (G, S) is easily solvable by a Turing machine over (G, S).

We already observed in section 2.1 that general tape graphs can have pathological word problems, but we could still hope for the situation to be better if we restrict to Cayley tapes. Unfortunately, it has long been known [2, 14] that there exist groups with undecidable word problems. So, this leads us to believe that Turing machines on a tape graph with unsolvable word problem are strictly more powerful than standard Turing machines. However, this requires that Turing machines over said graph also be able to compute all computable functions. Fortunately, this is the case.

3.1 Simulation of Standard Turing machines

Theorem III.1. *Let*

$$M = (Q_M, \Gamma_M, b_M, \Sigma_M, \delta_M, q_{0M}, F_M)$$

be a standard one-dimensional one-tape Turing Machine and let (G, S) *be a tape graph. Then, there is a Turing Machine over* (G, S) *that can simulate* M*.*

Here, a standard machine is a one-dimensional machine with a one-way infinite tape. For a technical definition, see Lecture 28 in [11].

The simulation itself is very straight-forward. The only difficulty stems from the question of how to arrange the tape contents of the simulated machine on the tape graph. If we could compute an infinite non-self-intersecting path through the tape graph, we could use this as a standard one-dimensional tape and do the simulation there. However, as we will see in Section 5.1, even in a Cayley tape, such a computable path need not exist.

Fortunately, we can do the simulation anyway, in this case, by a variant of the "always turn left" algorithm for solving mazes. By putting an ordering on the colors of our tape graph, "always turn left" becomes "always follow the lexicographically minimal edge". Thus, we can do the simulation on the tree constructed in Section 2.2.2 with the nth vertex in the well-ordering on the vertices of the tree storing the contents of the nth tape cell.

We will not give a fully detailed proof immediately, since the details are quite intricate and largely unenlightening. Instead, we will put the details off until Section 3.2. We will, however, describe the machine doing the simulating and explain how most of the pieces work.

Proof sketch. Let (G, S) be the tape graph given in the statement of the theorem. Denote the elements of S by $s_1, \ldots, s_n$. It will be convenient to extend S to a set $S' = S \cup \{s_0, s_{n+1}\}$ with the ordering $s_0 < s_1 < \cdots < s_n < s_{n+1}$. We will consider both s_0 and s_{n+1} to be colors that form a loop at each vertex. Define a Turing Machine, N, over (G, S) as follows:

- $Q_N = Q_M \sqcup (Q_M \times S') \sqcup (Q_M \times S') \sqcup (Q_M \times S) \sqcup (Q_M \times S)$
- $\Gamma_N = \Gamma_M \times S' \times \mathcal{P}(S) \times \mathcal{P}(S)$
- $b_N = (b_M, s_0, \emptyset, \emptyset)$
- $\Sigma_N = \Sigma_M$
- $q_{0N} = q_{0M}$
- $F_N = F_M \times S' \times \mathcal{P}(S) \times \mathcal{P}(S)$

Each symbol in the tape alphabet will have an intended meaning. Remember that we are going to do the computation on a tree, so we have to encode in each node not just the symbol of the simulated machine, but also auxiliary information about the structure of the tree. In particular, if $(\gamma, \sigma, A, B) \in \Gamma_N$, γ is the symbol of the simulated machine stored at the node, σ is the color to follow to reach the ancestor of this node in the tree, A is the set of colors corresponding to edges pointing away from the root in the tree, and B is the set of colors defining non-edges of the tree. Remember that we are going to be constructing this tree on the fly and so we don't have complete information about which colors correspond to edges of the tree at every step. Thus, elements of $S \backslash (A \cup B)$ are the edges whose membership in the tree has not yet been determined.

We will also give each state a name and an intended interpretation,

- Cq for $q \in Q_M$: We are simulating the computation of M and the current state of M is q.
- Rqx for $q \in Q_M$ and $x \in S'$: The simulated tape head is moving to the right and M is currently in state q. The argument x encodes the edge we followed to reach our current location.

- Lqx for $q \in Q_M$ and $x \in S'$: The simulated tape head is moving to the left and M is currently in state q. The argument x encodes the edge we followed to reach our current location.

- Eqx for $q \in Q_M$ and $x \in S$: The simulated tape head is moving to the right and M is in state q, but we have run out of tape and are attempting to extend the tree along edge x.

- Bqx for $q \in Q_M$ and $x \in S$: M is currently in state q and we just failed to extend the tree along edge x, so we are backtracking.

Note that the starting state of N is named Cq_{0_M}.

It will also be convenient to talk about the component functions of the transition function of M,

$$\begin{aligned} \delta_1 : Q_M \times \Gamma_M &\rightarrow Q_M \\ \delta_2 : Q_M \times \Gamma_M &\rightarrow \Gamma_M \\ \delta_3 : Q_M \times \Gamma_M &\rightarrow \{L, R\} \end{aligned}$$

We can now write down the action of the transition function of N. If the current symbol being read is (γ, σ, A, B), then the value of the transition function on each type of state is given in Table 3.1. Most of these transitions are pretty opaque, so some explanation is warranted.

If we are in a C-state, we perform one step of the computation of M and then transition into either an R-state or an L-state depending on whether the computation tries to move left or right. A leftward step in the simulated machine always begins with a step toward the root for the simulating machine, so we take that step immediately. Rightward steps are more complicated, so we leave the tape head where it is on a rightward step.

Table 3.1: Transition Table for N

$\digamma \in \Gamma_N$	$\delta_N(\digamma, (\gamma, \sigma, A, B))$
Cq	$\begin{cases} (R\delta_1(q,\gamma)s_0, (\delta_2(q,\gamma), \sigma, A, B), s_0) & \text{if } \delta_3(q,\gamma) = R \\ (L\delta_1(q,\gamma)\sigma^{-1}, (\delta_2(q,\gamma), \sigma, A, B), \sigma) & \text{if } \delta_3(q,\gamma) = L \end{cases}$
Lqx	$\begin{cases} (Cq, (\gamma, \sigma, A, B), s_0) & \text{if } \forall y \in A, y \geq x \\ (Lqs_{n+1}, (\gamma, \sigma, A, B), \max_{y \in A, y < x} y) & \text{otherwise} \end{cases}$
Rqx	$\begin{cases} (Cq, (\gamma, \sigma, A, B), \min_{y \in A, y > x} y) & \text{if } \exists y \in A, y > x \\ (Eqy, (\gamma, \sigma, A \cup \{y\}, B), y) & \text{where } y = \min_{z \in S \setminus (B \cup A), z > x} z \\ & \text{if } \exists z \in S \setminus (B \cup A), z > x \\ (Rq\sigma^{-1}, (\gamma, \sigma, A, B), \sigma) & \text{otherwise} \end{cases}$
Eqx	$\begin{cases} (Cq, (\gamma, x^{-1}, \emptyset, \{x^{-1}\}), s_0) & \text{if } \sigma = s_0 \text{ and } A = B = \emptyset \\ (Bqx, (\gamma, \sigma, A, B), x^{-1}) & \text{otherwise} \end{cases}$
Bqx	$(Rqx, (\gamma, \sigma, A \setminus \{x\}, B \cup \{x\}), s_0)$

Entries are triples in the order: new state, symbol written, direction tape head moves.

Taking a step to the left, we want to find the lexicographic immediate predecessor of our current vertex. We begin by taking one step toward the root, which we did when we transitioned out of the C-state. Then, either there is a branch all of whose elements are less than where we started, or not. If not, then we are at the immediate predecessor of our origin, and we continue computing. Otherwise, follow the greatest such branch and always move to the greatest child until we reach a dead-end. This dead-end is the immediate predecessor we were looking for, so we can continue the computation.

Taking a step to the right is significantly more complicated, as we are most likely going to have to extend the tree as we go. If we have already built some tree above us (A is non-empty), then we simply use that part of the tree, moving along the minimal edge in the tree and continuing the computation. This is the reason for using the "false" generator s_0 when we transition out of the C-state. Otherwise, try to extend the tree along the least edge that we have not already ruled out. We add that edge to A and switch to an E-state.

If A and B are both empty and $\sigma = s_0$, then this is a new vertex that we have not visited before, so continue with the computation. Otherwise, we are at a vertex that has already been added elsewhere in the tree. So, we back up and transition to a B-state. In the B-state, we rule out the edge we just took by removing it from A and adding it to B and switch to an R-state to try extending the tree again.

If we can't extend the tree at all ($B = \{s_1, \ldots, s_n\}$), then take a step toward the root and try again, eschewing anything less than or equal to the edge we backtracked along. Since there is an infinite tree for us to use, we will eventually be able to extend our construction of it and continue the computation. □

3.2 Proof of Theorem III.1

In this section, we give a rigorous proof of Theorem III.1. The proof is very technical and provides little intuition beyond what was given in the previous section. However, being familiar with the intuition is invaluable in following the technical development.

For the purpose of clarity, we restate the main theorem.

Theorem III.2. *Let*

$$M = (Q_M, \Gamma_M, b_M, \Sigma_M, \delta_M, q_{0M}, F_M) \tag{3.1}$$

be a standard one-dimensional one-way infinite one-tape Turing Machine and let (G, S) *be a tape graph. Then there is a Turing Machine over* (G, S) *that can simulate* M.

Our simulation will have difficulties if the tape head of the simulated machine falls off the left side of the tape, so we will assume that M never does this. Many people take this as part of the definition of a Turing Machine, but even if we do not, any program that allows for this can be easily converted to one that does not.

Notice that any transition from a C-state takes the control of N into a non-C-state. Also, note that the first component of each tape symbol (representing the symbol of the simulated machine at the corresponding location) and the q-component of the name of the state are changed only in a C-state. Thus, the computation naturally breaks into stretches between successive C-states. Since the simulation starts in a C-state, each of these stretches will begin with the control of N in a C-state and end with N transitioning into another C-state. Since the tree, R, is not given as part of the input, N will need to construct R on-line. So during these stretches, N will be using the auxiliary information stored at each tape cell to navigate R and, if necessary, extend its construction.

To be precise, let q_k denote the state of M and let m_k denote the location of the tape head of M after k steps. Since the simulation will take place entirely on the tree R and we have established that the lexicographic order on R is of order type ω, we can address tape cells of N with natural numbers. In particular, we will define a function, $addr : g \mapsto n$ if $g \in R \subseteq G$ and g is the nth element in the lexicographic ordering of R. It will also be convenient to define $R_k = \{v \in R | addr(v) \leq \max_{k'<k} m_{k'}\}$ under the convention that the maximum of the empty set is $-\infty$. Thus R_k includes all those vertices visited up to, but not including the simulation of the kth step of M. Notice that, in particular, R_k includes the location of the tape head of N at the beginning of the simulation of the kth step of M. Also, we will frequently need to refer to the greatest common ancestor, according to the tree order, of a pair of vertices in R. For h_1 and h_2 in R, we will denote their greatest common ancestor by $h_1 \wedge h_2$. Note that this is, in fact, the meet of these two elements in the meet-semi-lattice induced by the tree order on R.

Now, we will show three things. First, that the read and write behavior of N in

state Cq matches the behavior of M in state q. Second, that the machine begins in state Cq_0 and at address 0. Third, that if N is in state Cq_k at address m_k, then the next time N transfers into a C-state, it will be into state Cq_{k+1} at address m_{k+1}. This is enough to ensure that N faithfully reproduces the behavior of M. The first two of these claims will be treated in the computation step section. The final claim is the most difficult to show and will be treated in the leftward and rightward step sections.

To show that N transitions to a cell with the appropriate address before returning to a C-state, we will preserve a number of invariants between occurrences of C-states. Define the functions γ_g, σ_g, A_g, and B_g as the first, second, third, and fourth components respectively of the symbol written at vertex $g \in G$. Recall that γ_g is intended to represent the symbol stored by M in cell $addr(g)$, σ_g is a pointer to the immediate parent of g in R, A_g is a set of pointers to immediate descendants of g in R and B_g is a set of pointers to vertices that are adjacent to, but not immediate descendants of g in R. Our invariants are then that after k simulated steps of M, when N is in state Cq_k and the tape head of N is at vertex h_k with address m_k,

1. For all $g \in R_k \backslash \{e\}$, $g\sigma_g$ is the immediate ancestor of g in the tree order of R.

2. For all $g \in R_k$, for all $\alpha \in A_g$, $g\alpha$ is an immediate descendant of g in the tree order of R and if $g\delta \in R_k$ is an immediate descendant of g in the tree order, then $\delta \in A_g$.

3. For all $g \in R_k$, for all $\beta \in B_g$, $g\beta \in R_k$ and $g\beta$ is not an immediate descendant of g in the tree order.

4. For all $g \in R_k$, if $\alpha \in A_g$, $\beta < \alpha$, and $g\beta$ is an immediate descendant of g in the tree order, then $\beta \in A_g$

5. For all $g \in R_k$, for all $\alpha \in A_g$, $g\alpha \in R_k$.

6. For all $k \geq 1$, $g \in R_k$ if and only if $\sigma_g \neq s_0$ or $A_g \cup B_g \neq \emptyset$.

3.2.1 Computation Step

Notice that the first component of the tape symbol written in state Cq is $\delta_2(q, \gamma)$, which is exactly what M writes when in state q while reading the symbol γ. Also, the q-component of the new state is in all cases $\delta_1(q, \gamma)$, which is the state to which M would transition. Finally, though we will not need this immediately, N transitions into an R-state if M would move right and into an L-state if M would move left. This establishes the fact that the read/write behavior of N matches that of M.

As was observed before, the starting state of N is Cq_{0_M}. The starting location of the tape head of N is at the root of R, which is the minimal element in the lexicographic order since the lexicographic order agrees with the tree order on tree-comparable elements. Thus, the starting address of N is 0, the same as M. Also, $R_0 = \emptyset$ and $R_1 = \{e\}$ so all invariants are vacuously true when the machine starts.

As promised, this establishes the first two claims. For the third claim, there are two cases, depending on whether the tape head of M goes left or right at time k.

3.2.2 Leftward Step

First, suppose that the tape head moves left. Then we know that N transitions into state $Lq_{k+1}\sigma_{h_k}^{-1}$. In addition, the tape head moves to $h_k\sigma_{h_k}$. We know that $h_k \neq e$ since M never takes a leftward step from the leftmost cell on its tape. Thus, by invariant (1), $h_k\sigma_{h_k}$ is the immediate ancestor of h_k in the tree order.

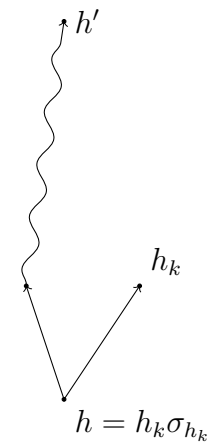
It is enough to prove the following two statements:

- Every vertex visited before the next occurrence of a C-state is an ancestor of (or simply is) the vertex with address $m_{k+1} = m_k - 1$.

- At each step, the head moves to an immediate descendant of the current vertex and remains in an L-state unless the address of the current vertex is m_{k+1}, in which case it stays put and transitions into a C-state.

Let $h' = addr^{-1}(m_k - 1)$ and let $h = h' \wedge h_k$. Note that since the lexicographic order agrees with the tree order whenever both are defined, $addr^{-1}(m_k) = h_k$ is not an ancestor of $addr^{-1}(m_{k+1}) = addr^{-1}(m_k - 1)$. Thus, $h_k\sigma_{h_k}$ must be a descendant of h. Suppose, toward a contradiction, that $h_k\sigma_{h_k}$ was a proper descendant of h. Then there would exist α and β such that $h\alpha$ was an immediate descendant of h and an ancestor of h' and $h\beta$ was an immediate descendant of h and an ancestor of h_k (thus also an ancestor of $h_k\sigma_{h_k}$). By the definition of the ordering, since $addr(h') < addr(h_k)$, $\alpha < \beta$. However, since $h\beta$ was also an ancestor of $h_k\sigma_{h_k}$, we would also have that $addr(h') < addr(h_k\sigma_{h_k})$. Appealing to the tree order, we get $m_k - 1 = addr(h') < addr(h_k\sigma_{h_k}) < addr(h_k) = m_k$. You cannot find an integer between successive integers, so this is a contradiction and in fact, $h_k\sigma_{h_k} = h$. In particular, this means that the first statement is satisfied immediately upon entering an L-state.

Figure 3.1: Portion of the tree during a leftward step



Straight arrows indicate edges, while squiggly arrows indicate paths of length ≥ 0.

The first step taken in an L-state is slightly different than subsequent steps, so we treat the first step first. In the first case, suppose $addr(h) = m_{k+1}$. We know that $\sigma_{h_k}^{-1} \in A_h$ by (2) since $h\sigma_{h_k}^{-1} = h_k$ is an immediate descendant of h in R_k. Supposing, then, that $\delta \in A_h$ with $\delta < \sigma_{h_k}^{-1}$, we have

$$m_k - 1 = m_{k+1} = addr(h) < addr(h\delta) < addr(h\sigma_{h_k}^{-1}) = addr(h_k) = m_k \quad .$$

This is a contradiction, so N transitions into the state Cq and the tape head does not move. Both of our statements to be proved are trivial in this case.

In the second case, suppose $addr(h) < m_{k+1}$. Let $h\alpha$ be the immediate descendant of h that is an ancestor of h'. Then $\alpha < \sigma_{h_k}^{-1}$ since otherwise,

$$m_k - 1 = addr(h') \geq addr(h_k) = m_k$$

Also, if $\delta \in A_h$ with $\alpha < \delta < \sigma_{h_k}^{-1}$, then

$$m_k - 1 = addr(h') < addr(h\delta) < addr(h_k) = m_k \quad ,$$

again a contradiction, so N transitions to Lqs_{n+1} and the tape head moves to $h\alpha$, writing nothing on the tape. The head has moved to a descendant of h in the tree, hence an ancestor of h', so both statements that we are trying to prove are again true.

Now we can consider all steps beyond the first. Again, we separate into two cases depending on the address of the current vertex. Let us call the current vertex g and start by supposing that $addr(g) = m_{k+1}$. Since we are past the first step, N is in state Lqs_{n+1}, so $A_g \neq \emptyset$ is equivalent to there being some element of A_g that is less than s_{n+1}. Suppose, toward a contradiction, that $\delta \in A_g$. Then $g\delta$ is an immediate descendant of g by (2) and we have

$$m_k - 1 = addr(g) < addr(g\delta) < addr(h_k) = m_k \quad .$$

This is a contradiction, so no element of A_g is less than s_{n+1} and so N transitions into state Cq.

Next, suppose that $addr(g) < m_{k+1}$. Define $g\alpha$ to be the immediate descendant of g which is also an ancestor of h'. Notice that $\alpha \in A_g$ again by (2) since $g \in R_k$. Suppose toward a contradiction that $\delta \in A_g$ with $\delta > \alpha$. Then, by (2) again, $g\delta$ is

an immediate descendant of g and

$$m_k - 1 = addr(h') < addr(g\delta) < addr(h_k) = m_k \quad ,$$

again a contradiction, so N remains in state Lqs_{n+1} and the tape head moves to $g\alpha$. Since $g\alpha$ is a descendant of g in the tree and is an ancestor of h, our inductive hypotheses continue to be true.

Hence, the sequence of vertices visited during this stretch of L-states is a path through the tree starting at $h_k\sigma_{h_k}$ and terminating at the vertex with address m_{k+1}. Note also that nothing has been written on the tape during the entire process. Thus, since $m_{k+1} < m_k$ and therefore $R_{k+1} = R_k$, the invariants 1-6 about tree structure continue to hold.

3.2.3 Rightward Step

Rightward steps are by far the most complicated, primarily because it is during rightward steps that the construction of R occurs. Each rightward step breaks down into segments between successive R-states just as the larger computation breaks down into segments between successive C-states. So, we introduce some extra invariants to be conserved between R-states. Define $h = addr^{-1}(m_k) \wedge addr^{-1}(m_{k+1})$, call the current state Rqx, and call the current cell g. Our new invariants are then:

7. The tape head is at a descendant of h.

8. If gy, $y \in (S \cup \{s_0\})\backslash\{\sigma_g\}$, is an ancestor of cell m_k, then $y \leq x$.

9. If the tape head is at h and gz, $z \in S\backslash\{\sigma_h\}$, is an ancestor of cell m_{k+1}, then $x < z$.

10. With y as in (8), if $y < w \leq x$, then gw is not an immediate descendant of g in the tree order.

When the machine first enters the R-state, $x = s_0$ and $addr(g) = m_k$. Thus, $gs_0 = g = gs_0 \wedge g$ is trivially an ancestor of cell m_k. Thus, in this instantiation of (8), $y = s_0$ and $y \leq x$. Whether the tape head is already at h or not, $x = s_0$ is minimal so we automatically get $x \leq z$. Also, the tape head trivially begins at a descendant of h. Now we only need to verify that N preserves these invariants between R-states.

There will be four cases, depending on whether the tape head is or is not at h and the whether the sets $\{z \in A_g | z > x\}$ and $F = \{z \in S \backslash (B_g \cup A_g) | z > x\}$ are or are not empty.

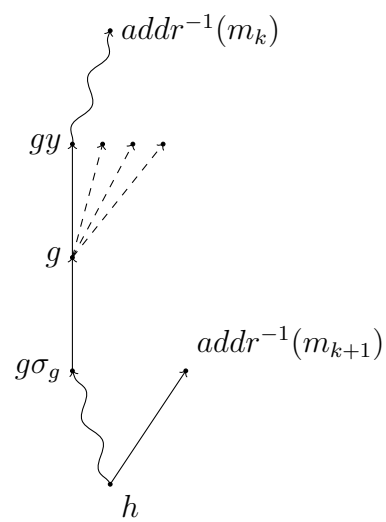
The tape head is not at h and F is empty

Since the tape head is not at h, it is at some proper descendant of h. Let y be as in (8). If there is some $\sigma \in A_g$ with $\sigma > x$, then $m_k < addr(g\sigma)$ since g is an ancestor of cell m_k and $\sigma > x \geq y$. But g is a strict descendant of h, so we get the chain of inequalities,

$$m_k < addr(g\sigma) < m_{k+1} = m_k + 1$$

So, $\{\sigma \in A_g | \sigma > x\} = \emptyset$ and N does not transition to a C-state. Also, since F is empty, there are no elements of $S \backslash B_g$ greater than x, so N does not transition into an E-state either.

Figure 3.2: Portion of the tree during a rightward step when the tape head is not at h and F is empty



Straight arrows indicate edges, while squiggly arrows indicate paths of length ≥ 0. Dotted arrows indicate edges in B_g.

This means that we must be in the "otherwise" case and so the tape head of N moves to cell $g\sigma_g$, which by (1) is the immediate ancestor of g. Since g was a proper descendant of h, the immediate ancestor of g is also a descendant of h. Finally, N transitions into state $Rq\sigma_g^{-1}$.

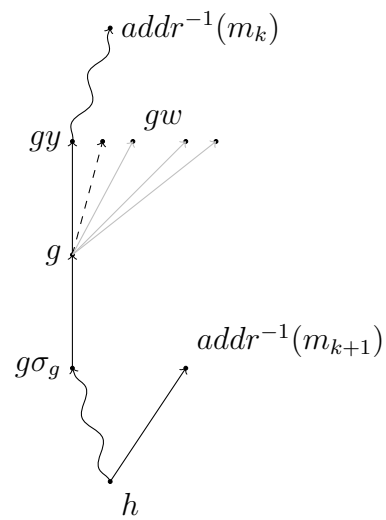
This means that the x argument now points to $g\sigma_g\sigma_g^{-1} = g$, which is an ancestor of cell m_k. Thus, $x = \sigma_g^{-1}$ satisfies the properties of y in (8) and so trivially, $y \leq x$ and (10) holds. Also, gx is an ancestor of cell m_k and gz, with z as given by (9), is an ancestor of cell m_{k+1}. Thus, by definition of lexicographic ordering, $x < z$ so we preserve (9) as well.

Notice that in this case, we have not altered σ, A, or B at any cell, nor have we visited a cell outside of R_k.

The tape head is not at h and F is not empty

By an argument identical to the previous case, N does not transition into a C-state. However, by definition, N does transition into an E-state. In particular, if we let w be the minimal element of F, N moves to gw, transitions to state Eqw and adds w to the set A_g. In effect, N is assuming that w points to an immediate descendant of g in the tree and will act accordingly until it finds out differently.

Figure 3.3: Portion of the tree during a rightward step when the tape head is not at h and F is not empty



Straight arrows indicate edges, while squiggly arrows indicate paths of length ≥ 0. Dotted arrows indicate edges in B_g and gray edges indicate edges in F.

Now, since the tape head did not begin at h, gw cannot be an immediate descendant of g in the tree. Otherwise, we would have $m_k < addr(gw) < m_{k+1}$ by the same argument we have used above. The only way gw can not be an immediate descendant is if gw is already in R_k. Thus, by (6), $\sigma_{gw} \neq s_0$ or $A_{gw} \cup B_{gw} \neq \emptyset$. In either case, N does not enter a C-state. Instead, N enters state Bqw and moves to cell $gww^{-1} = g$. Once back at g, the tape head removes w from A_g (it found out differently), puts it into B_g, and transitions into

state Rqw.

Since the tape head has returned to cell g, it is at a descendant of h by (7). Given y as in (8), $y < x < w$ by definition of w, so (8) is preserved as well. By assumption, $g \neq h$, so (9) is satisfied trivially. Let $y < w' \leq w$. If $w' < w$, then by the minimality of w, $w' \in A_g \cup B_g$, but since N is not transitioning to a C-state, $w' \notin A_g$. Thus $w' \in B_g$ and by (3), gw' is not an immediate descendant of g. Also, if $w' = w$, then $gw' = gw$ is not an immediate descendant of g as we have already shown. This shows that (10) still holds.

The total change to the tree-structure data in this case is that w is put into B_g. As mentioned, gw is not an immediate descendant of g, so the truth of invariant (3) is unchanged. The only cell visited, gw, is already in R_k so R_k does not grow and the remaining invariants are preserved.

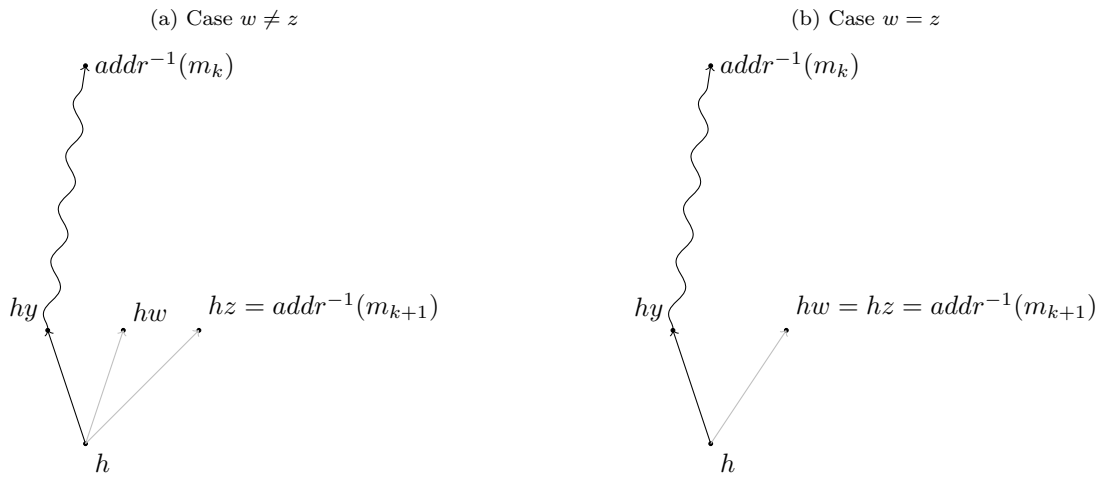
Notice that this case can only occur at most $|S|$ times before $S \backslash B_g$ is exhausted, at which point the first case must apply and we must move to the immediate ancestor of g. Thus, it is at most $|S|$ steps before we move to the immediate ancestor again. Repeating, we must eventually reach h. This brings us to the last two cases.

Tape head is at h and A_h has no elements greater than x

Let y and z be as in (8) and (9) respectively. By (9), we know that $x < z$. Thus, $z \notin A_h$. Since hz is an immediate descendant of h, (3) implies that $z \notin B_h$. Hence, $z \in S \backslash (A_h \cup B_h)$. Define w to be the minimal element of $S \backslash (A_h \cup B_h)$ greater than x. Then, N transitions to state Eqw, adds w to A_h, and moves to hw.

We now have two subcases. In the first case, $w \neq z$. Then $w < z$ and it is impossible that hw is an immediate descendant of h. Otherwise, we would have $m_k < addr(hw) < addr(hz) \leq m_k + 1$. Since hw is not an immediate descendant of h, the cell hw must already be in R_k. So, by (6), $\sigma_{hw} \neq s_0$ or $A_{hw} \cup B_{hw} \neq \emptyset$.

Figure 3.4: Portion of the tree during a rightward step when the tape head is at h and A_h has no elements greater than x



Straight arrows indicate edges, while squiggly arrows indicate paths of length ≥ 0. Dotted arrows indicate edges in B_g.

Hence, N transitions to state Bqw and moves the tape head to $hww^{-1} = h$. Then N removes w from A_h, puts it in B_h, and transitions to state Rqw without moving the tape head. Notice that if we find ourselves in this case again, the value of w will have increased, so eventually we will reach the second subcase where $w = z$.

In the first subcase, the tape head has not moved, so (7) is preserved. By construction, $w > x \geq y$ so (8) is also preserved. By the minimality of w, $w < z$, so (9) is also preserved. Suppose $y < w' \leq w$. If $w' \leq x$, then we can apply (10). Otherwise, if $w' < w$, by the minimality of w, $w' \in A_h \cup B_h$. But A_h has no elements greater than x, so $w' \in B_h$. By (3), hw' is not an immediate descendant of h. If $w' = w$, then we have already shown that hw is not an immediate descendant of h, establishing (10). Again, the only tree structure data that has changed is that w was added to B_h. It was already observed that hw is not an immediate descendant of h, so invariant (3) is preserved.

For the second subcase, suppose $w = z$. By the contrapositive of (2) and the

fact that $z \notin A_h$, $hz \notin R_k$. That means that, according to (6), $\sigma_{hz} = s_0$ and $A_{hz} = B_{hz} = \emptyset$. Thus, N transitions to Cq, records z^{-1} as σ_{hz}, adds z^{-1} to B_{hz}, and leaves the tape head where it is. That location is hz, a descendant of cell m_{k+1}. We have the chain of inequalities, $m_k < addr(hz) \leq m_{k+1} = m_k + 1$, so in fact, $addr(hz) = m_{k+1}$, as needed.

In this subcase, N transitions to a C-state so we only need to check that (1)-(6) are preserved. The only tree structure data that has changed here is that z was added to A_h, z^{-1} was added to B_{hz}, and σ_{hz} was set to z^{-1}. In addition, hz was visited for the first time, adding it to R_{k+1}.

All of the invariants are potentially affected in this case, so we will treat them each individually. Many will make use of the fact that hz is an immediate descendant of h, which we have previously shown.

- Invariant (1): For all $g \in R_k$, we simply appeal to (1) itself. For hz, notice that $hz\sigma_{hz} = hzz^{-1} = h$, which is clearly the immediate ancestor of hz.

- Invariant (2): For all $g \in R_k \backslash \{h\}$, we can again appeal to (2). For h, z was added to A_h, but hz is an immediate descendant of h, so the first part of (2) is still true. The second part is also true because all immediate descendants of h in R_{k+1} are either in R_k and we can apply (2) or are hz and $z \in A_h$.

- Invariant (3): For all $g \in R_k$, apply (3). For hz, $B_{hz} = \{z^{-1}\}$. $hzz^{-1} = h \in R_k$ is not an immediate descendant of hz, so (3) follows.

- Invariant (4): For all $g \in R_k \backslash \{h\}$, apply (4). Let $\alpha \in A_h$, and $\beta < \alpha$ with $\beta \notin A_h$. The only element of A_h that is greater than x is z $(= w)$. Thus, $\beta < z = w$. If $\beta \leq y$, then we can apply (4) to conclude that $h\beta$ is not an immediate descendant of h. Otherwise, $y < \beta < w$, so we can apply (10) to

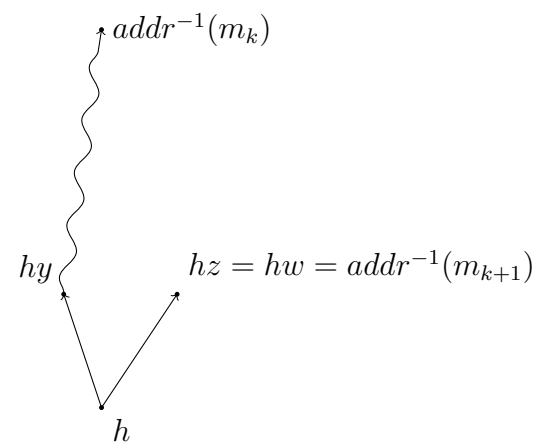
conclude the same. By contraposition, (4) is true at h. A_{hz} is empty so (4) holds trivially there.

- Invariant (5): For all $g \in R_k$, apply (5). Observe that $A_{hz} = \emptyset$.
- Invariant (6): If $k = 0$, then $R_k = \{e, z\}$. $z \in A_e$, $\sigma_z = z^{-1}$, and all other cells are blank, so (6) holds. If $k > 0$, then apply (6) to all elements of R_k. In addition, $\sigma_{hz} = z^{-1}$, so (6) holds here as well.

Tape head is at h and A_h has some element greater than x

Let y and z be as given by (8) and (9) and let w be the minimal element of A_h that is greater than x. If $w < z$, then $m_k < addr(hw) < addr(hz) \leq m_k + 1$, which is impossible. If $w > z$, then by (4), $z \in A_h$. But $z > x$ by (9) and w was picked to be the minimal element of A_h greater than x. Thus, we have a contradiction so we must have $w = z$.

Figure 3.5: Portion of the tree during a rightward step when the tape head is at h and A_h has some element greater than x



Straight arrows indicate edges, while squiggly arrows indicate paths of length ≥ 0.

Following the transition table of N, N transitions to state Cq and moves to vertex $hw = hz$. Since $z > y$, $m_k < addr(hz) \leq m_k + 1 = m_{k+1}$. Thus, the tape head is now at cell m_{k+1}.

Again, none of the tree structure data has changed and by (5), $hz \in R_k$, so R_k does not grow either. $\square$

3.3 Keeping Track of Memory

We turn now from the task of simulating garden-variety Turing machines to other typical foundational tasks; in particular, the problem of keeping track of memory use by a computation. This is a non-trivial task, but without it, some very intuitive algorithms fail.

For example, we alluded earlier to the fact that a Turing machine over G can easily solve the word problem for G. We now state that fact officially as a lemma.

Lemma III.3. *Let (G, S) be a tape graph. There is a Turing machine over (G, S) that can solve the word problem for G.*

For the purpose of this lemma, it is easier to think of the input provided, not on a tree, as described in section 2.2.2, but as a stream of symbols from S. Equivalently, the input could be provided on a read-only, one-way, one-dimensional tape. In the future, we will be interested in using the word problem for G as an oracle, so our input stream will be the sequence of symbols written on an oracle tape, which is why we will use the input stream formalism.

The simple proof goes exactly as you would expect. Mark the starting position of the read/write head. As you read symbols from the input stream, the read/write head follows the corresponding edge. At the end of the input, if the symbol read by the read/write head is the starting marker, accept. Otherwise, reject. This suffices to prove the theorem as stated. However, this isn't sufficient for using the word problem as an oracle. The reason is that this process cannot be iterated. Upon completion of this algorithm, there is a starting marker somewhere on the tape and the read/write head is (probably) somewhere else on the tape. In order for the machine to reset to run this algorithm again, the tape head must return to its starting position. Since

the input was read-once, the tape head must do an exhaustive search of the entire tape to find its starting location and without the ability to keep track of memory use, it is not obvious that this can be done.

We replace this naive algorithm with a slightly less naive approach in which the tape head leaves a "trail of breadcrumbs" that it can use to return to its starting location. Again, if we want to iterate the algorithm, the tape head is going to need to clean up this trail upon completion. This slightly less naive approach writes at each tape cell visited a pointer to the last tape cell visited. This way, at the end of the input stream, the oracle head can return to its point of origin, erasing its pointers along the way. However, this also does not work. It is possible that this trail of pointers may cross itself at some point. The first pointer written there will be overwritten, breaking the chain of pointers back to the start.

Instead, we will build a spanning tree for the portion of the tape used during the computation, rooted at the starting vertex. This is clearly sufficient for using the tape as an oracle for the word problem, since we can execute the naive algorithm and when we are done, erase the tree one leaf at a time. Thus, the lemma is really a corollary of the following fact.

Fact III.4. *Given some machine,* $M = (Q_M, \Gamma_M, b_M, \Sigma_M, \delta_M, q_{0M}, F_M)$*, over* (G, S)*, there is some machine,* N *that performs the same computation as* M*, but also constructs a spanning tree for the portion of tape that* M *has used*

The construction of this tree will go in much the same manner that it does in the proof of Theorem III.1. There is a slight complication in that the simulated machine can move in any direction, rather than just left or right, but there is also a major simplification in that the spanning tree we construct does not need to have any association with any externally defined order.

Proof. As in the proof of Theorem III.1, we start with a definition of N.

- $Q_N = Q_M \times (\{W\} \sqcup S \sqcup S)$
- $\Gamma_N = \Gamma_M \times \mathcal{P}(S) \times (S \cup \{-\})$
- $b_N = (b_M, \emptyset, -)$
- $\Sigma_N = \Sigma_M$
- $F_N = F_M$

A symbol of (γ, A, σ) written on the tape will have the intended interpretation that γ is the symbol written on M's tape at this location, A is the collection of pointers to descendants of the current cell in T, and σ is a pointer to the ancestor of the current cell in T with the symbol $-$ reserved for the root. We also inherit the definitions of δ_1, δ_2, and δ_3 as the component functions of M's transition function. Again, as in the proof of Theorem III.1, we then give the elements of $\{W\} \sqcup S \sqcup S$ names and describe their intended interpretations.

- W: N is waiting for M's tape head to move.
- V_s for $s \in S$: M's tape head has just moved in direction s and N is verifying that this has not caused the tape head to loop back to a previously visited cell.
- E_s for $s \in S$: M's tape head moved in direction s and the verification failed (the head has looped back). We are in the process of erasing the last edge we added to the tree.

The start state of N will be the state named (q_{0M}, W). We will also call the tree being constructed by N, T.

All that is left to be done is to describe the transition table of N (Table 3.2).

Table 3.2: Transition Table for N

$F \in \Gamma_N$	$\delta_N(F, (\gamma, A, \sigma))$
(q, W)	$\begin{cases} ((\delta_1(q,\gamma), W), (\delta_2(q,\gamma), A, \sigma), \sigma), \delta_3(q,\gamma)) \text{ if } \delta_3(q,\gamma) \in A \\ ((\delta_1(q,\gamma), V_{\delta_3(q,\gamma)}), (\delta_2(q,\gamma), A \cup \{\delta_3(q,\gamma)\}, \sigma), \delta_3(q,\gamma)) \text{ otherwise} \end{cases}$
(q, V_s)	$\begin{cases} ((q, W), (\gamma, A, s^{-1}), 0) \text{ if } \sigma = - \text{ and } A = \emptyset \\ ((q, E_s), (\gamma, A, \sigma), s^{-1}) \text{ otherwise} \end{cases}$
(q, E_s)	$((q, W), (\gamma, A \backslash \{s\}, \sigma), s)$

Entries are triples in the order: new state, symbol written, direction tape head moves.

The W-states of this machine serve the same role as the C-states of the machine in the proof of Theorem III.1. In the same way as in that proof, the simulated state and the symbol written on the simulated tape are only changed in W-states. The changes all come directly from the transition function for M, so the only things to check are that the tape head moves to where it is supposed to move to and that the construction of the spanning tree works. Fortunately, stretches between W-states are much simpler than stretches between C-states in the proof of Theorem III.1.

We deal first with the movement of the tape head. There are only three possible state sequences between W-states. The first is $W \to W$, the second is $W \to V \to W$, and the third is $W \to V \to E \to W$. In the first case, the tape head simply moves in the direction $\delta_3(q, \gamma)$, just as M's head does. In the second case, the simulated tape head moves first in the direction $\delta_3(q, \gamma)$, then not at all. Thus, the tape head of N ends up where M's does. In the third case, the simulated tape head moves first in the direction $\delta_3(q, \gamma)$, then in the direction $\delta_3(q, \gamma)^{-1}$, then again in direction $\delta_3(q, \gamma)$. Thus, the total motion is $\delta_3(q, \gamma)$, the same as for M.

To show that the spanning tree is constructed correctly, we introduce some definitions and invariants. First, we give the starting vertex the name v_0 and then we

define

$$R_k = \{v_0\} \cup \{g | \sigma_g \neq - \text{ or } A_g \neq \emptyset\}$$

after N has simulated M for k steps.

We also define a tree structure on R_k by giving σ_g and A_g the usual interpretations, namely that σ_g points to the immediate ancestor of g in the tree (when $g \neq v_0$) and A_g is a collection of pointers to the immediate descendants of g in the tree.

Our invariants are,

1. R_k is a tree
2. If M has visited vertex g by step k, then $g \in R_k$
3. For all $k > 0$, if $\sigma_g = -$ and $A_g = \emptyset$, then $g \notin R_k$

Lastly, we assume that the tape head of M always moves.

When the machine first starts, N has simulated 0 steps of M and $R_k = \{v_0\}$. Clearly, all three invariants are satisfied. Suppose, then, that all invariants are satisfied after N has simulated k steps of M. Suppose also that N starts in state (q, W) and is reading the symbol $(\gamma_g, A_g, \sigma_g)$ at tape cell g. Call $\delta_1(q, \gamma_g)$ q' and call $\delta_3(q, \gamma_g)$ s. Then there are three cases.

The first case is when N transitions directly from W-state to W-state. In this case, $s \in A_g$. Since R_k is a tree, that means that gs is an immediate descendant of g in R_k. In particular, this means that $k > 0$. None of the data describing the tree has been changed, so 1 and 3 continue to hold. Also, the new vertex visited, gs, is already in R_k, so 2 continues to hold as well.

The second case is when N transitions to a V-state and then right back to a W-state. Then N will add s to A_g, move to gs and write s^{-1} to σ_{gs}. Thus, only the vertex gs is added to R_k to give R_{k+1}.

We start by showing that R_{k+1} is a tree. Since $s \in A_g$, gs should be an immediate descendant of g in the tree. This agrees with the fact that $s^{-1} = \sigma_{gs}$, so the only thing left to check is that gs was not already a node in the tree. Since N transitioned from a V-state directly back to a W-state, $\sigma_{gs} = -$ and $A_{gs} = \emptyset$. Therefore, by invariant 3, either $g \notin R_k$ and we are done or $k = 0$. If $k = 0$, $R_k = \{v_0\} = \{g\}$. Since the tape head of M always moves, $gs \neq g$ so $g \notin R_k$.

Second, we have to show that if M has visited vertex g by step $k + 1$, then $g \in R_{k+1}$. By invariant 2, we only have to consider the vertex visited on step $k + 1$, which is gs by the definition of s. We have already observed that this is in R_{k+1}.

Third, we must ensure that if $\sigma_h = -$ and $A_h = \emptyset$, then $h \notin R_{k+1}$. Again, the only tape cells to be written on were g and gs. So, if $\sigma_h = -$ and $A_h = \emptyset$, then either $h \notin R_k$ or $k = 0$ by invariant 3. If $k = 0$, then $R_k = \{v_0\} = \{g\}$ and $R_{k+1} = \{g, gs\}$. Both g and gs were written on, so $h \neq g$ and $h \neq gs$ so $h \notin R_{k+1}$. On the other hand, if $h \notin R_k$, then we still have that $h \neq gs$ since gs was written on, so since $R_{k+1} = R_k \cup \{gs\}$, $h \notin R_{k+1}$.

The third case is when N transitions to a V-state, then to an E-state, and then back to a W-state. Then N will add s to A_g, move to gs, write nothing, return to g, remove s from A_g, and finally move back to gs. Since we are not in the first case, when N goes directly from W-state to W-state, s was not originally in A_g and so the net effect of this is that the tape head of N moves from g to gs and nothing else changes.

Here, $R_{k+1} = R_k$, so R_{k+1} is clearly a tree. Also, by virtue of transitioning into an E-state. we know that either $\sigma_{gs} \neq -$ or $A_g \neq \emptyset$. Since these both correspond to the blank tape symbol, the tape head of N must have been at vertex gs sometime before. In particular, $k \neq 0$. Thus, since $R_{k+1} = R_k$, invariant 3 continues to hold.

The only invariant of interest in this second case is invariant 2. We know that M visits cell gs on step $k+1$, but gs is not added to the tree in step $k+1$. So, we need to ensure that $gs \in R_k$. However, remember that $\sigma_{gs} \neq -$ or $A_{gs} \neq \emptyset$. Thus, $gs \in R_k$ by definition. $\square$

The fact that we can build this spanning tree is much more powerful than just resetting the tape after a computation. With such a tree, a machine can exhaustively search through all cells visited so far in the course of a computation. This means, for example, that the usual proof that multiple heads do not increase computational power goes through directly to machines over arbitrary tape graphs. The locations of all the heads can simply be marked on the tape with special symbols and for each step of the simulated machine, the simulating machine can search through the entire used tape for all the head symbols and then update the tape accordingly. Similarly, a machine with multiple, identical tapes can be simulated by a machine with a single tape of the same type by enlarging the tape alphabet to tuples of tape symbols together with special symbols for the head on each tape and follow the procedure for multiple heads.

3.4 An Alternative Characterization of the Turing Degrees

We have demonstrated that Turing Machines on arbitrary tape graphs are strictly more powerful than standard Turing Machines, so the next question to ask is, "how much more powerful?" The short answer is "as powerful as we want". If we allow tape graphs in full generality, then, as was observed in section 2.1, we can encode any language we want into the word problem of the tape. The more interesting question is whether the same thing is true if we restrict our tape graphs to Cayley tapes.

In [3, 4] Boone showed that for any r.e. Turing degree, there is a finitely presented

group whose word problem is in that degree. Using such a group, we can produce a machine (more precisely a class of machines) that computes exactly the functions in or below the given degree. More precisely,

Theorem III.5. *Let T be an r.e. Turing degree. There is a finitely generated group, (G, S), such that the class of functions computable by a Turing Machine over G is exactly the class of functions in or below T.*

Proof. By Boone, let G be a group whose word problem is in T and let f be a function in or below T. Since $f \leq T$, f is Turing reducible to the word problem for G. Turing machines over G can perform this reduction since they can compute all recursive functions and they can solve the word problem for G by Lemma III.3. Therefore, they can compute f.

To show that a Turing Machine over G cannot compute any language that is not in or below T, observe that a Turing Machine with an oracle for the word problem for G can easily simulate a Turing Machine over G. It simply maintains a list of nodes written as a sequence of generators for the address together with whichever tape symbol is written there. When the simulated tape head moves, the machine simply appends the generator to the current address and consults the oracle to determine which node in the list this address corresponds to, adding a new entry if the new node isn't in the list. □

3.4.1 Further Remarks

At the end of the previous section, we described how the usual results about multiple tapes and multiple heads go through to the setting of tape graphs. We left out the case where the machine has multiple non-identical tapes. This case does not have an analog in the world of standard Turing machines, as people are largely

uninterested in machines that have access to separate tapes of different dimensions.

Over tape graphs, however, there is potentially something to be gained by having access to multiple, different tapes. In this case, the class of functions computable by the machine is the class of functions in or below the join of the r.e. degrees of the word problems of the tapes. Just as in Section 3.4, this machine is mutually simulatable with a standard Turing machine with oracles for the word problem of each tape.

CHAPTER IV

Efficiency of Turing Machines on Tape Graphs

4.1 Computational Complexity over Cayley Graphs

Up to this point, we have been unconcerned with questions of efficiency. However, different models of computation, while defining equivalent notions of computable function, can define dramatically different notions of efficient computation. As it turns out, we get much the same flavor of results in the world of complexity that we did in the world of computability. This follows essentially from the fact that the simulation given in Theorem III.1 is efficient.

To make sense of the efficiency of Turing machines over tape graphs, we first need to be clear about what we mean by the time and space use of a Turing machine over a tape graph. The notion of time is exactly the same in this context, but space is not quite so simple. In the usual one-dimensional context, there are two ways to define the space use of a computation. The first and most common is to define the space use as the total number of squares visited during the course of the computation. The second, and the way that we will use, is to define space use as the largest number of non-blank cells at any one time.

Even in the one-dimensional case, these notions are inequivalent. Clearly, the number of non-blank tape cells can never exceed the maximum of the length of

the input and the number of cells visited, but one could encode a long sequence of numbers in unary as the number of blanks between successive marked cells. Thus, the definition of space in terms of non-blank cells allows machines that are at least as powerful as those allowed by the standard definition. They are not dramatically more powerful, as sequences of blanks could be encoded in binary, rather than unary, which would increase the space use by no more than a factor logarithmic in the run time.

4.2 Efficiency of Theorem III.1

Theorem IV.1. *Suppose that M is a garden-variety Turing machine with access to an oracle for the word problem for G and denote by $S(n)$ the space use and by $T(n)$ the time use of M on inputs of length n and by $R(n)$ the number of times the tape head reverses direction on inputs of length n. Then for any tape graph, (G, S), there is a 2-head machine, N with given tape graph that simulates M and runs in time*

$$6((S(n)-1)R(n) + |S|S(n)) + 5T(n) = \mathcal{O}(T(n)^2) \tag{4.1}$$

and space $S(n)$.

In addition, if N is any machine over (G, S), running in time $T(n)$ and space $S(n)$, then there is a multi-tape Turing machine, M, that simulates N in time $\mathcal{O}(T(n)^4)$ and space $\mathcal{O}(S(n)^2)$ when given an oracle for $w(G)$.

From this, we get the corollary,

Corollary IV.2. *The class of functions computable in polynomial time on a Turing machine over a graph, G, is exactly $\mathsf{P}^{w(G)}$ and the class of functions computable in polynomial space is $\mathsf{PSPACE}^{w(G)}$.*

Theorem IV.1 follows from the proof of Theorem III.1 by observing that the

simulation given there is efficient and that if a garden-variety Turing machine is given access to an oracle for $w(G)$, it can efficiently simulate the action of any machine over G.

At this point, we encounter an ambiguity well known in the world of complexity theory. In the statement of Theorem IV.1, when we say that the oracle machine M runs in space $S(n)$, how does that restrict the length of queries that M can make to the oracle? There are two choices. Either the oracle tape counts toward the space bound, in which case the longest query that M can make is size $S(n)$ or else the oracle tape does not count and the longest query that M can make is of length $2^{O(S(n))}$. Theorem IV.1 is true only if we take the first meaning, namely that the oracle tape counts toward the space bound. Thinking the other way around, this is another piece of evidence suggesting that counting the oracle tape toward the space bound is the "correct" measure of space use of an oracle machine.

4.3 Proof of Theorem IV.1

4.3.1 Machines over G can simulate machines with oracles

One head of N is going to represent the read/write tape head of M and the other is going to represent the write head on the oracle tape. We will talk about these tape heads as if they are two separate one-head machines with identical tapes and free communication. However, they may as well be a single machine with a single tape by the usual argument (see Lecture 30 in [11]). The read/write head will act just as in the proof of Theorem III.1, while the action of the oracle head will be described later.

It was observed in the proof of Theorem III.1 that the simulation naturally decomposes into stretches between C-states. For the purposes of efficient exposition, we ignore the C-states and once we do, the simulation naturally decomposes into

stretches of contiguous L-states and stretches of everything else. These stretches correspond to leftward and rightward sweeps respectively of the simulated machine. Since C-states never occur back-to-back, we have thrown out at most half the steps taken by the simulating machine by ignoring the C-states.

Leftward sweep

Let us consider first a stretch of contiguous L-states. Since the simulated machine never leaves the left end of the tape, we don't have to worry about the simulating tape head leaving the tree R. We will count the steps taken by the simulating machine in this right-to-left sweep by counting the number of times each edge of R is traversed.

First, we identify each edge, e, of R with the endpoint of e furthest from the root of R. Since R is a tree, this is a bijection between non-root vertices of R and edges of R. We will denote this bijection by $v \mapsto e_v$. Notice that the two endpoints of e_v are v and $v\sigma_v$. We argue that for each edge, e_v, there is at most one time during the sweep when the simulating tape head is at v and follows edge e_v.

By inspecting the transition table, we see that the only time the simulating tape head moves closer to the root is when transitioning out of a C-state into an L-state. Therefore, unless the simulating machine is in a C-state at v, it does not leave along edge e_v. But if the simulating machine is in a C-state at v, the simulated machine's tape head is at cell $addr(v)$. This can happen at most once during one leftward sweep of the simulated machine.

R is a tree and so the removal of any edge disconnects R as a graph. Together with the fact that each edge is traversed at most once in the root-ward direction, this means that each edge is traversed at most three times in total. Otherwise, three of those traversals would be away from the root and two of those traversals would be contiguous. In order for that to happen, the tape head would have to get from

v to $v\sigma_v$ without using edge e_v, which can't happen since v and $v\sigma_v$ are in different connected components of $R \backslash e_v$.

The number of vertices in R is bounded by $S(n)$ so the number of steps taken in a leftward sweep is bounded by $3(S(n) - 1)$.

Rightward sweep

On a rightward sweep, we must contend with the fact that R may need to be extended, possibly several times. Nevertheless, the argument will parallel that for a leftward sweep fairly closely. In fact, we will make use of the same bijection between edges and non-root vertices.

This time, however, we will show that for each edge, e_v, there is at most one time during the sweep when the simulating tape head is at $v\sigma_v$ and follows edge e_v. This argument is slightly more complicated in this case, since at the time that the simulating tape head arrives at $v\sigma_v$, vertex v may not yet be part of R. Thus, if the simulating tape head is at $v\sigma_v$, then there are two different conditions under which it will follow edge e_v.

In the first case, the simulating machine transitions into a C-state. In this case, the simulated tape head is at cell $addr(v)$, which happens at most once during a rightward sweep. In the second case, the simulating machine transitions into an E-state. Since the machine is trying to extend R, v is not yet in R. Therefore, the simulating machine then transitions into a C-state. Again, this means that the simulated machine's tape head is at $addr(v)$, which happens at most once during a rightward sweep.

Just as in the leftward case, each edge of R can be traversed at most three times, so again the number of steps taken by the simulating machine along edges in R in a rightward sweep is bounded by $3(S(n) - 1)$. This says nothing about the number of

steps taken along edges that are not in R.

Steps along edges that are not in R

Any time the simulating tape head moves from a vertex, v, to $v\sigma_v$ or vx for some $x \in A_v$, the head is following an edge of R. The only time the tape head does not do that is when it transitions into a B- or E-state. The only way to enter a B-state is from an E-state, so considering the case of entering an E-state is sufficient. So, suppose that the simulating tape head is at vertex v and transitions to state Eqy, adds y to A_v, and moves to vertex vy. Then one of two things happens. Either $\sigma_{vy} = s_0$ and $A_{vy} = B_{vy} = \emptyset$, in which case the machine transitions to a C-state. In this case, vy is an immediate descendant of v in R, so the edge (v, vy) is in R. Since we are only counting steps along edges not in R, we discount this step.

On the other hand, if $\sigma_{vy} \neq g_0$ or one of A_{vy} and B_{vy} is non-empty, vy is already in R and therefore not an immediate descendant of v. Then the simulating machine follows edge y^{-1} and transitions to state Bqy. From there, the machine removes y from A_v and adds it to B_v. The net effect of this is that the simulating machine has taken three steps and added one element to B_v. Therefore, the total contribution of steps along edges that are not in R is bounded by $3\sum_v |B_v|$ where the sum is taken over all vertices visited during the computation. This is clearly bounded by $3|S|S(n)$.

Action of the oracle head

The oracle head is responsible for simulating calls made by M to the oracle for the word problem for G. That this can be done at all was proved in section 3.3, so we just need to show that it can be done efficiently.

Recall that for each query, the naive algorithm for solving the word problem takes

one step for each symbol written on the query tape. We are simultaneously building a spanning tree, so are interested in how many steps the machine from section 3.3 takes simulating a query of length L. Steps that extend the spanning tree take two steps (the state sequence is W, V, and back to W) while all the rest take three steps (the state sequence is W, V, E, then back to W) for a total of at most $3L$ steps. To reset the tape head and erase the tree then takes at most another $2L$ steps. So all-in-all, $5L$ steps are taken.

The sum of the lengths of all queries to the oracle cannot exceed the time use of the machine, so when we calculate the total time taken to simulate all the oracle queries, we get the bound of $5T(n)$.

Space bound

The space bound is much easier to show. Notice that every vertex visited by the tape head of the simulating machine is either already in R or is added as soon as the tape head moves there. Thus, the number of cells used by the simulating machine is exactly the same as the number used by the simulated machine. All that remains is to measure the space used in simulating oracle calls.

The naive algorithm for making an oracle call to the word problem uses at most one cell for each symbol in the query. After returning the result of the query, all marks made are erased, so we don't have to worry about accumulating space use between oracle calls. Since the length of the query counts toward the space bound of the simulated machine, calls to the oracle also remain within the space bound.

4.3.2 Machines with oracles can simulate machines over G

Here, M is a two-tape Turing machine that intends to simulate N, a machine over the tape graph, (G, S). Programming multi-tape Turing machines is a more familiar

task, so we won't write out a full transition table, but instead describe the algorithm. It is the same algorithm as the one sketched at the end of section 3.4.

M will store the tape of N as a sequence of address, data pairs written sequentially on one tape. For each tape cell used by N, there will be one entry in the list from the set $S^* \times \Gamma_M$. The first component will give a path from the starting vertex of N and the second component will give the symbol written at the endpoint of that path.

To do the simulation, M will maintain an address of the current location of N's tape head on the second tape. Each step will consist of the tape head on M's first tape rewinding to the beginning, reading each address on the first tape one at a time, querying the oracle about whether it is equal to the address on the second tape, and when it finds a match, reading the symbol from the corresponding address-data pair, and consulting the description of N to determine which symbol to append to the address on the second tape (the direction N moves), which simulated state to transition to, and which symbol to write on the first tape. In the case that it finds no match, it assumes the simulated tape contains a blank, transitions accordingly, and copies the contents of the second tape to the first as a new entry in the list.

The time complexity of this is $T(n)^2$ steps to rewind the second tape, $T(n)$ steps to read and query the oracle about each of $T(n)$ addresses and possibly another $T(n)$ steps to create a new list entry. Altogether, that makes $\mathcal{O}(T(n)^2)$ steps. $\square$

4.4 Connections with Combinatorial Group Theory

Equipped with Theorem IV.1, we can now relate the existence and non-existence of groups with interesting properties to statements about computational complexity.

4.4.1 A Polynomial-time analog of the Boone-Novikov Theorem

Remember that in section 3.4, we used a result of Boone and Novikov to show that arbitrary r.e. Turing degrees can be realized as the class of functions computable on Turing Machines over some group. It would be nice to have a comparable result about the P-degrees. This immediately presents the following question:

Question IV.3. *Is there a word problem in every* P*-degree?*

This is exactly the polynomial time analog of the result of Boone and Novikov mentioned earlier. As far as we have been able to determine, this is still open, but there are partial results in this direction. Both Stillwell [19] and Sapir, Birget, and Rips [16] construct groups which can efficiently simulate Turing machines. This means that for any language, L, there is a group, G, such that $L \leq^P w(G)$ and the word problem of G can be solved in time polynomial in the time complexity of L. Thus the word problem for G is *hard* for the P-degree of L, but it may not be complete. Stillwell also makes the point that his construction works in linear space.

Many interesting complexity classes are defined by a time or space bound and have complete problems. To make interesting statements about these classes, the results above are sufficient. For example,

Remark IV.4. There is a group over which P and PSPACE are identical.

Consider the group, G, given by Stillwell's construction in [19] beginning with some polynomial space machine deciding TQBF, the language of totally quantified boolean formulas. There is a P-time reduction from TQBF to the word problem over G by the construction, so $w(G)$ is PSPACE-hard. Also by the construction, the space complexity of $w(G)$ is only linearly worse than the space complexity of the naive algorithm for TQBF, thus polynomial. Together, this means that $w(G)$ is

PSPACE-complete.

Now, over that G,

$$\mathsf{PSPACE}^{w(G)} = \mathsf{PSPACE} = \mathsf{P}^{w(G)}$$

The first equality follows from the fact that $w(G)$ is in PSPACE and the second from the fact that any language in PSPACE is P-time reducible to $w(G)$ since $w(G)$ is PSPACE-complete.

Really all we have done here is taken a known relativization result [1] and encoded the associated oracle into the tape geometry. However, with a positive solution to question IV.3, we would be able to take *all* relativization results about polynomial time computation and encode the oracle into the tape. This suggests that non-relativizing class separations will be very difficult to establish based on a nuts-and-bolts of Turing machines argument. Any such argument must make essential use of the structure of the tape.

4.4.2 Nearly Linear Time

A common critique of Turing machines in complexity theory is that they are rather far removed from the functioning of a real computer. Of course, up to polynomial time, they are equivalent, but for lower-level complexity measures, they are not equivalent. In fact, the notion of linear time isn't even the same on one- and two-headed Turing machines.

Gurevich and Shelah introduced a more robust notion than linear time, called *nearly-linear time*, denoted NLT [9]. This is the class of problems solvable in time $n\mathsf{polylog}(n)$ on a RAM model. They showed that this class was robust under nearly any choice of machine model. The notable exception was multi-tape Turing machines. In fact, the class of problems solvable in $n\mathsf{polylog}(n)$ time on multi-tape Turing

machines was introduced much earlier by Schnorr in [17] and is called *quasi-linear time*, denoted QLT. It remains open whether $\mathsf{QLT} = \mathsf{NLT}$ [13].

Despite the fact that multi-tape Turing machines give a potentially different notion of nearly-linear time from RAM models, random access Turing machines do not. What is more, random access Turing machines can be simulated in nearly-linear time by Turing machines over an appropriate group. That group will be a variant on the Baumslag-Solitar group, $BS(1,2)$, but first we define what Gurevich and Shelah mean by a random-access Turing machine.

Definition IV.5. A *random-access Turing machine* is a Turing machine with three one-dimensional tapes. This machine functions just like a standard three-tape machine except that the tape head on the first tape (also the input tape), called the main tape, cannot be moved directly. Instead, it is always at the cell whose address is written in binary on the second tape, called the address tape.

Notice that changing the bit in the kth cell of the address tape causes the tape head on the main tape to move a distance of 2^k cells. This would take far too long on a standard three-tape machine. Instead, we use a group with built-in shortcuts.

The Baumslag-Solitar group, $BS(1,2)$ is the group $\langle a, b | aba^{-1} = b^2 \rangle$. In this group, b is an element of infinite order, so we use the tape cells $\{b^k\}_{k\in\mathbb{Z}}$ to simulate all three tapes of the random access machine. Note that we will still use three heads; we will simply superimpose the contents of all three tapes. Now, since $a^k b a^{-k} = b^{2^k}$, changing a bit in the kth cell of the simulated address tape only requires the head on the simulated main tape to move $2k+1$ steps, a vast improvement.

However, this improvement is still not enough. In a computation of length T, the bits near address $T/2$ could be flipped roughly $T/2$ times. Each flip requires the head on the simulated main tape to move roughly T times for a total run time of

about $T^2/2$, which is not nearly-linear.

The fix is to use the same trick again. Introduce another generator, c, the relation, $cac^{-1} = a^2$, and another tape head. This tape head keeps track of the address of the head on the simulated address tape. Doing so requires only a multiplicative $\log(n\mathsf{polylog}(n)) = \mathsf{polylog}(n)$ overhead. Now, when the kth bit is flipped on the simulated address tape, the tape head on the simulated main tape follows the path,

$$(c^{|k|-1}a^{k_{|k|}}c^{-1}a^{k_{|k|-1}}c^{-1}\dots c^{-1}a^{k_0})b(c^{|k|-1}a^{k_{|k|}}c^{-1}a^{k_{|k|-1}}c^{-1}\dots c^{-1}a^{k_0})^{-1} = a^k b a^{-k} = b^{2^k}$$

where $|k|$ is the length of k written in binary and k_i is the ith bit of k with $k_{|k|}$ being the high-order bit. Thus, the main head makes at most $6\log k + 1$ steps. Since k is nearly-linear in n, the whole simulation is done in nearly-linear time.

What we end up with is a multi-head machine that is combinatorially tractable in the way that Turing machines are, but has computational power essentially equivalent to much more realistic RAM models.

4.4.3 Groups of Polynomial Growth

Tape graphs have a natural structure as a metric space under the usual graph metric. This means that we can talk about the ball of radius r about a vertex, v, $B_r(v)$. In the case of Cayley tapes, this is identical to the word metric. Since a Cayley graph is vertex transitive, the graphs $B_r(v)$ are isomorphic for each choice of v. This allows us to define a notion of the rate of growth of a group as the asymptotic growth rate of the number of vertices in $B_r(v)$. In particular, we will be interested in groups for which $B_r(v)$ is bounded by a polynomial in r, called groups of polynomial growth.

The reason we are interested in the growth rate of groups is that in any computation of length T, every vertex visited in the course of the computation is contained

in $B_T(e)$. This means that we can use the usual simulation of Turing machines by circuits to produce circuits simulating Turing machines over tape graphs (see Lecture 6 in [12]). This allows us to establish statements such as,

Remark IV.6. If G is a group with polynomial growth and L is computable in polynomial time over G, then $L \in \mathsf{P/poly}$.

Since G is of polynomial growth, only polynomially many tape cells can be visited during a polynomial time computation. That section of tape can then be encoded in a circuit of polynomial size in the usual way. It would be interesting to know whether this goes the other way.

Question IV.7. *If $L \in \mathsf{P/poly}$, is there some group of polynomial growth, G, over which L is polynomial time computable?*

In general, if the word problem for G is sufficiently hard, say NP-complete, then we can take L to be NP-complete.

Remark IV.8. If a group of polynomial growth has NP-complete word problem, then $\mathsf{NP} \subset \mathsf{P/poly}$.

Since $\mathsf{NP} \subset \mathsf{P/poly}$ is widely believed to be false, this suggests that it is unlikely that groups of polynomial growth can have NP-complete word problems. Some evidence of this exists already. For example, Gromov's theorem on groups of polynomial growth says that such groups must have a nilpotent subgroup of finite index. Nilpotent groups are known to have word problems in P, so it seems likely that groups with polynomial growth should have easily solvable word problems.

CHAPTER V

Inescapable Groups

5.1 What is an Inescapable Group?

Theorem III.1 raised the question of whether or not a Turing Machine can always walk off to infinity on the Cayley graph of an infinite group without retracing its steps. This suggests the following definition:

Definition V.1. An *inescapable group* is a Cayley graph, (G, S), such that any infinite computable sequence, s, of elements from S corresponds to a self-intersecting path.

This definition leads to a number of questions. Do such things exist? What do they look like? Is inescapability a group invariant? Very little is known in response. One thing that is known, as we will see in Section 5.3, is that an inescapable group must be a Burnside group; that is, every element must have finite order. However, first we address the more fundamental question of whether or not inescapable groups even exist.

5.2 Construction of an Inescapable Group

Definitions

We will need to make a few definitions. Let A be a finite set. Let $A^* = A^{<\omega}$ be the set of all finite sequences with elements from A, called words over A. Recall that

λ denotes the empty word and let $A^+ = A^* \backslash \{\lambda\}$ be the set of non-empty words. If $w \in A^*$, denote by $|w|$ the length of w, by $w(i)$ the ith symbol in w (indexed from 0), and by $w(i,j)$ the word $w(i)w(i+1)\ldots w(j)$. Also, let $A^{\leq n} = \{w \in A^*, |w| \leq n\}$ be the set of words of length no greater than n.

For $w, w' \in A^*$ define the following relations:

- w' is a subword of w if there exist $0 \leq i, j < |w|$ such that $w' = w(i,j)$.
- w' is a subsequence of w if there exist $0 \leq i_1 < i_2 < \ldots < i_{|w'|} < |w|$ such that $w' = w(i_1)w(i_2)\ldots w(i_{|w'|})$.
- $\#(w, w') = \left|\{x \in \mathbb{N}^{|w'|} | x_0 < \ldots < x_{|w'|-1} \text{ and } w(x_i) = w'(i) \text{ for all } i\}\right|$ is the number of ways in which w' is a subsequence of w.

Notice that λ is both a subword and subsequence of every word and that $\#(w, \lambda) = 1$ for all w. As an example, note that while *aabbaa* does not contain *aba* as a subword, it does contain *aba* as a subsequence in 8 different ways.

We will say that an infinite sequence of symbols from A, call it s, is computable if there is a Turing Machine which, on input n, produces $s(n)$.

When dealing with sets, we will use $\mathcal{P}(X)$ to denote the collection of all subsets of X and $\mathcal{P}_r(X)$ to denote the set of all subsets of X of cardinality r.

We will also be dealing with polynomials in a non-commutative polynomial ring, so let $f \in K\langle x_1, \ldots, x_d\rangle$ be a polynomial over a field, K, with non-commuting indeterminates. We say that f is homogeneous of degree n if f is a K-linear combination of monomials of the form

$$x_{i_1}^{n_1} x_{i_2}^{n_2} \cdots x_{i_k}^{n_k}$$

with $n_1 + n_2 + \ldots + n_k = n$. In this case, we write $\partial(f) = n$.

5.2.1 A Combinatorial Lemma

When we get to the actual construction, we are going to need to find a subsequence of each computable sequence that satisfies certain properties. In particular, we need the following lemma.

Lemma V.2. *Let $A = \{x_1, \ldots, x_m\}$ and let $n \geq 1$. There is some $C(m,n)$ such that for all words, s, over alphabet A with $|s| \geq C(m,n)$, s contains a non-empty subword, w, with the following property. For each $w' \in A^{\leq n} \cap A^+$, w contains w' as a subsequence an even number of times.*

In fact, the number $C(m,n)$ is a Ramsey number, $R(2,3,2^{\frac{m^{n+1}-1}{m-1}})$ where the function $R(r,k,n)$ is given in Ramsey's Theorem,

Theorem V.3 (Ramsey)**.** *Let r,k,n be positive integers with $1 \leq r \leq k$. Then there exists an integer, denoted $R(r,k,n)$, such that for each set X with $|X| = R(r,k,n)$ and each partition of $\mathcal{P}_r(X)$, $Y_1, \ldots, Y_n$, there exists a k-element subset Y of X and a set Y_i with $\mathcal{P}_r(Y) \subset Y_i$.*

To prove the lemma, we need a theorem from the combinatorial theory of words.

Theorem V.4 (Pirillo [15])**.** *Let $\phi : A^+ \to E$ be a mapping from A^+ to a set E with $|E| = n$. For each $k \geq 1$, each word $w \in A^+$ of length $R(2,k+1,n)$ contains a subword $w_1 w_2 \ldots w_k$ with $w_i \in A^+$ and*

$$\phi(w_i w_{i+1} \ldots w_{i'}) = \phi(w_j w_{j+1} \ldots w_{j'}))$$

for all pairs (i,i'), (j,j') ($1 \leq i \leq i' \leq k$ and $1 \leq j \leq j' \leq k$).

Proof of Lemma V.2. Consider the function $\phi : A^+ \to \mathbb{Z}_2^{A^{\leq n}}$ defined as follows:

$$(\phi(w))\,(w') = \#(w,w') \bmod 2$$

Since

$$\left|\mathbb{Z}_2^{A^{\leq n}}\right| = 2^{\frac{m^{n+1}-1}{m-1}}$$

We can apply Theorem V.4 with $k = 2$ to s to get w_1w_2, a subword of s such that

$$\phi(w_1) = \phi(w_2) = \phi(w_1w_2)$$

Then the word w_1w_2 contains any non-empty sequence over A of length no greater than n as a subsequence an even number of times.

We will prove something slightly stronger by induction on the length of the contained subsequence. Specifically, we will show that not only does w_1w_2 satisfy the theorem, but so do w_1 and w_2 individually.

For the base case, let x_i be a subsequence of length one. Then

$$\phi(w_1w_2)(x_i) = \phi(w_1)(x_i) + \phi(w_2)(x_i) = 2\phi(w_1)(x_i) = 0 \bmod 2$$

since the number of occurrences of a single symbol in a concatenation of words is simply the sum of the number of occurrences in each factor. Using the fact that $\phi(w_1w_2) = \phi(w_1) = \phi(w_2)$, we have established the base.

The induction step isn't significantly more difficult; the only difficulty arises from the fact that the number of occurrences of a substring of length greater than one isn't additive. However, we do have the formula,

$$\#(w_1w_2, w') = \sum_{i=0}^{|w'|} \#(w_1, w'(0, i-1))\#(w_2, w'(i, |w'|-1))$$

We really only care about the parity of this expression and if we assume the induction hypothesis for all strings shorter than w', most of the terms are even. So by reducing,

$$\#(w_1w_2, w') = \#(w_1, w') + \#(w_2, w') \bmod 2$$

Now, in an argument analogous to the base case, we get

$$\phi(w_1w_2) = \phi(w_1) = \phi(w_2) = 0 \bmod 2 \qquad \square$$

5.2.2 The Golod-Shafarevich Theorem

The Golod-Shafarevich Theorem is a powerful tool from algebra that gives a sufficient condition for a particular quotient algebra to be infinite dimensional. For our purposes, it is a tool that will ensure that as we add relations to a free group, we don't collapse the group to something finite.

The power of the theorem comes from the fact that the criterion it presents is based only on the number of relations of certain types, and not on the relations themselves. This gives us a large amount of freedom to choose the relations we want without having to worry about bad interactions between them.

Theorem V.5 (Golod-Shafarevich [7]). *Let $R_d = K\langle x_1, \ldots, x_d\rangle$ be the polynomial ring over a field, K, in the non-commuting indeterminates $x_1, \ldots, x_d$. Let $f_1, f_2, \ldots$ be a set of homogeneous polynomials of R_d, and let the number of polynomials of degree i be r_i. Let $2 \leq \partial(f_i) \leq \partial(f_{i+1})$ and let I be the ideal generated by $f_1, f_2, \ldots$. Let $R_d/I = A$. If all the coefficients in the power series,*

$$\left(1 - dt + \sum_{i=2}^{\infty} r_i t^i\right)^{-1}$$

are non-negative, then A is infinite dimensional.

In a subsequent paper, Golod [6] proves the following corollary,

Corollary V.6. *In Theorem V.5, if*

$$r_i \leq \epsilon^2 (d - 2\epsilon)^{i-2}$$

where $0 < \epsilon < \frac{d}{2}$, then A is infinite-dimensional.

For example, taking $d = 2$ and $\epsilon = \frac{1}{4}$ in the corollary, we see that if $r_i \leq 2$ for all $i \geq 11$ and $r_i = 0$ for all $i < 11$, A is infinite dimensional.

Golod used this fact to establish the existence of a Burnside group, that is, an infinite group in which every element has finite order. He did more than this, in fact, and produced an infinite p-group for each prime, p. The diagonalization in Section 5.2.3 will follow the same general ideas as Golod's construction as simplified for countable fields in Fischer and Struik [5].

5.2.3 The Construction

In Fischer and Struik [5], there is a construction of a nil-algebra over finite and countable fields. Although not expressly stated there, the construction is essentially a diagonalization over all polynomials. In fact, even Golod's original construction of a nil-algebra can be viewed as a diagonalization over all polynomials, but the fact that the collection of all polynomials over an arbitrary field is in general uncountable makes it more difficult to see.

The presentation given here will follow the construction in Fischer and Struik, since we will be diagonalizing against the set of computable sequences, which is countable. Thus, we will use the more straightforward construction.

Theorem V.7. *There exists an inescapable group.*

Proof. Let $d \geq 2$ and $A = \{x_1, \ldots, x_d\}$. Consider the algebra, $\mathcal{A} = \mathbb{F}_2\langle A\rangle$ in non-commuting indeterminates, $x_i \in A$. Let

$$S = \bigcup_{i=1}^{d} \{(1 + x_i), (1 + x_i)^{15}\}$$

and enumerate all r.e. sequences over S: $s_0, s_1, \ldots$. We will use the elements of S interchangeably as characters in an alphabet and as polynomials in $\mathcal{A}$.

We will construct a set of homogeneous polynomials, F, such that the following conditions hold:

- $x_i^{16} \in F$ for $1 \leq i \leq d$. (This will ensure that $(1 + x_i)$ has order 16 and is therefore multiplicatively invertible in the quotient algebra)

- For each $i \in \mathbb{N}$, s_i has a subword, w such that
$$\left(\prod_{j=0}^{|w|-1} w(j)\right) - 1$$
is in the ideal generated by elements of F.

- The number of elements of F of degree i is 0 for $i < 16$, d for $i = 16$, and either 0 or 1 for every $i > 16$.

Define the following sequence recursively:

$$\begin{aligned} r_0 &= 16 \\ r_{n+1} &= 15 \cdot R\left(2, 3, 2^{\frac{|S|^{r_n+1}-1}{|S|-1}}\right) \end{aligned}$$

We can now enumerate the elements of F as follows.

Start with $F = \{x_1^{16}, \ldots, x_d^{16}\}$ and begin enumerating the elements of all s_i in parallel. If, at any point, we have enumerated a contiguous subsequence of elements of some s_i of length r_{i+1}, call it v and do the following.

By Lemma V.2, we can find a non-empty subword, w, of v such that $|w| \leq \frac{1}{15} r_{i+1}$ and every sequence of length $\leq r_i$ occurs an even number of times in w. Then, by multiplying out,

$$p(x_1, \ldots, x_d) := \prod_{k=0}^{|w|-1} w(k) = \sum_{w' \in S^*} \#(w, w') \prod_{k=0}^{|w'|-1} (w'(k) - 1)$$

Note that if $0 < |w'| \leq r_i$, $\#(w, w') \equiv 0 \bmod 2$ and that the constant term of p is 1. Note also that $w'(k) - 1$ always has constant term 0, so $p - 1$ has no non-zero terms with degree $\leq r_i$. Since p is a polynomial, we can write $p-1$ as a sum of homogeneous components, $f_1, \ldots, f_m$. Note that for all $1 \leq i \leq m$, $r_i < \partial(f_i) \leq 15|w| \leq r_{i+1}$. Add

all of these to F, stop enumerating s_i but continue enumerating everything else that hasn't been similarly halted.

The F that is so constructed clearly contains x_i^{16} for $1 \leq i \leq d$ and for each s_i, if s_i is total, it has a subword whose corresponding product is equivalent to 1 modulo the ideal generated by F. Also, each s_i only adds polynomials to F that have degree in the interval $(r_i, r_{i+1}]$ and adds at most one polynomial of any given degree. Since we start with d polynomials of degree 16 and $r_0 = 16$, the number of elements of F of degree i must be 0 for all $i < 16$ and either 0 or 1 for every $i > 16$.

We would then like to show that the multiplicative semigroup of $\mathcal{A}/(F)$ generated by elements of S, call it G, is an inescapable group. First, note that by the binomial theorem,

$$(1 + x_i)(1 + x_i)^{15} = 1 + x_i^{16} = 1$$

so G is a genuine group and S is closed under inverses. S trivially generates G, so we only need G to be infinite for (G, S) to be a tape graph.

By elementary calculus,

$$2 \leq d \leq \frac{1}{16}(d - .5)^{i-2}$$

for all $i \geq 11$. Therefore, F satisfies the hypotheses of Corollary V.6 and $\mathcal{A}/(F)$ is infinite-dimensional.

Now, for any positive integer, d, there must be a monomial of degree d that does not lie in the ideal generated by F. Otherwise, every monomial of degree $\geq d$ would be in the ideal and the quotient algebra would be finite-dimensional. So, consider two generic such monomials,

$$x_{i_1} x_{i_2} \ldots x_{i_M} \text{ and } x_{j_1} x_{j_2} \ldots x_{j_N}$$

with $M > N$ and the group elements,

$$u = (1 + x_{i_1})(1 + x_{i_2}) \dots (1 + x_{i_M}) \text{ and } v = (1 + x_{j_1})(1 + x_{j_2}) \dots (1 + x_{j_N})$$

Then,

$$u - v = x_{i_1} x_{i_2} \dots x_{i_M} + \dots$$

where the remaining terms all have degree $\leq M$. Since the ideal (F) is generated only by homogeneous polynomials, $u - v \in (F)$ if and only if every homogeneous component of $u - v$ is in (F). However, the degree M component of $u - v$ is clearly not in (F), so neither is $u - v$. Therefore, u and v are different elements in G.

Thus, we can find infinitely many distinct elements of G, one for each degree. Therefore, G is infinite and (G, S) is a tape graph.

In addition, any computable sequence of generators is one of the s_i, so we have ensured that it has a subword such that the corresponding product is equal to 1 in the quotient algebra. This is the same as having product 1 in the group, so all computable sequences of generators must correspond to self-intersecting paths. $\square$

The construction given above in fact does better than producing an inescapable group. Since every step of the construction can be done recursively, the set of relations in the group is r.e. It is a standard result that a group with an r.e. set of relations is recursively presentable, so the construction produces a *recursively presentable* inescapable group. That being said, the question of whether there exists a finitely presentable inescapable group remains open. It is also unlikely that the word problem for the group constructed above is solvable, so there also remains the question of whether there exists an inescapable group with solvable word problem. It should also be noted that the only property of computable sequences that the construction used was that there are countably many of them. Thus, the argument

relativizes. In particular, for any Turing degree, T, the construction will produce a group with no escape in T, but with a presentation in T.

Since the presentation is in T, the set of escapes is Π^0_1 in T. You can see this by observing that if an infinite sequence has a self-intersection, we will eventually know about it. Trivially, there are escapes, so by the low basis theorem, there must be a low escape. So, for any Turing degree, we have a group with no escapes in or below the given degree, but with an escape low relative to it.

5.3 Inescapable Groups are Burnside Groups

It is clear that in an inescapable group, all generators must have finite order, but must every group element have finite order? An infinite group in which every element has finite order is called a Burnside group and the existence of Burnside groups was an open problem for some time before E. Golod constructed one in 1964 [7, 6].

So our question can be rephrased as, must every inescapable group be a Burnside group? It turns out the answer is yes, but it's not as obvious as it appears. Suppose your group did have an element of infinite order. The obvious thing to do would be to write this element of infinite order as a product of generators and simply repeat that sequence to produce an escape. This certainly gives a computable sequence of generators that hits infinitely many elements of the group, but there is no reason this is a *non-self-intersecting* path. Fortunately, we can find some other element of infinite order and an expression of it as a product of generators such that this naive construction does give a non-self-intersecting path.

The general idea is to start with the obvious construction and cut out the loops. Since our original element has infinite order, the constructed walk can only return to a given point a fixed, finite number of times. So, we wait at each point of the walk

until it returns to our current position for the last time, then we follow for one step, and repeat. The only difficulty is knowing when the last return will be. Fortunately, this is invariant under shifting by our infinite order element, so we can just record it in a finite table indexed by the generators in the expression of our infinite order element.

Theorem V.8. *Let $G = \langle g_0, \ldots, g_n \rangle$ be a group and $a \in G$ have infinite order. Then there exists $b = \prod_{j=0}^{k-1} g_{i_j} \in G$ such that $\prod_{j=0}^{N} g_{i_{j \bmod k}}$ is distinct for each N.*

Proof. Let $a = \prod_{i=0}^{m-1} h_i$ with $h_i \in \{g_0, \ldots, g_n\}$ be an expression for a of minimal length. Define

$$\delta(r, s, M) = \left(\prod_{i=r+1}^{m-1} h_i \right) a^M \left(\prod_{j=0}^{s-1} h_j \right)$$

and consider relations of the form $\delta(r, s, M) = e$ with $M \geq 0$. Fixing r and s, there is at most one M for which $\delta(r, s, M) = e$ since a has infinite order. Similarly, fixing r and M, there is at most one s for which $\delta(r, s, M) = e$ since we chose an expression for a of minimal length. This allows us to define the following functions, $\alpha : [0, m-1] \to \mathcal{P}(\mathbb{N} \times [0, m-1])$, $\beta : [0, m-1] \to \mathbb{N}$, and $\gamma : [0, m-1] \to [0, m-1]$ by

$$\begin{aligned}
\alpha(r) &= \{(M, s) | \delta(r, s, M) = e\} \\
\beta(r) &= \begin{cases} 0 \text{ if } \alpha(r) = \emptyset \\ \max_{(M,s) \in \alpha(r)} M \text{ otherwise} \end{cases} \\
\gamma(r) &= \begin{cases} s \text{ if } (\beta(r), s) \in \alpha(r) \\ (r+1) \bmod m \text{ if } \alpha(r) = \emptyset \end{cases}
\end{aligned}$$

Notice that $\alpha(r)$ is always a finite set since s can take on at most finitely many values and for each value, there is at most one M such that $(M, s) \in \alpha(r)$. Therefore,

whenever $\alpha(r)$ is non-empty, $\beta(r)$ exists and is attained by exactly one element, $(\beta(r), s')$ of $\alpha(r)$. By definition, $\gamma(r) = s'$, so γ is well-defined.

Lemma V.9. *For all n, there is a k_n such that*

$$\prod_{i=1}^{n} h_{\gamma^{(i)}(m-1)} = a^{k_n} h_0 \dots h_{\gamma^{(n)}(m-1)}$$

where the sequence $\{k_n\}$ is defined by

$$\begin{aligned} k_0 &= -1 \\ k_{n+1} &= k_n + \begin{cases} 0 \text{ if } \alpha(\gamma^{(n)}(m-1)) = \emptyset \text{ and } \gamma^{(n)}(m-1) \neq m-1 \\ 1 \text{ if } \alpha(\gamma^{(n)}(m-1)) = \emptyset \text{ and } \gamma^{(n)}(m-1) = m-1 \\ \beta(\gamma^{(n)}(m-1)) + 1 \text{ otherwise} \end{cases} \end{aligned}$$

Proof. We proceed by induction. If $n = 0$, the product on the left side is empty, so we can take $k_0 = -1$.

For the induction step, suppose the lemma holds for $n - 1$. Then we can apply the induction hypothesis to reduce the problem to

$$a^{k_{n-1}} h_0 \dots h_{\gamma^{(n-1)}(m-1)} h_{\gamma^{(n)}(m-1)} = a^{k_n} h_0 \dots h_{\gamma^{(n)}(m-1)}$$

or more simply,

$$h_0 \dots h_{\gamma^{(n-1)}(m-1)} = a^{k_n - k_{n-1}} h_0 \dots h_{\gamma^{(n)}(m-1)-1}$$

If $\alpha(\gamma^{(n-1)}(m-1))$ is empty, then

$$\gamma^{(n)}(m-1) = (\gamma^{(n-1)}(m-1) + 1) \bmod m$$

If $\gamma^{(n-1)}(m-1) = m-1$, then the product on the left is a and the product on the right is $a^{k_n - k_{n-1}}$, so take $k_n = k_{n-1} + 1$. Otherwise, take $k_n = k_{n-1}$.

On the other hand, if $\alpha(\gamma^{(n-1)}(m-1))$ is non-empty, then $\gamma^{(n)}(m-1) = s$ for $(M, s) \in \alpha(\gamma^{(n-1)}(m-1))$ where $M = \beta(\gamma^{(n)}(m-1))$. Since $(M, s) \in \alpha(\gamma^{(n-1)}(m-1))$,

$$\begin{aligned} h_0 \dots h_{\gamma^{(n-1)}(m-1)} &= h_0 \dots h_{\gamma^{(n-1)}(m-1)} \delta(\gamma^{(n-1)}(m-1), s, M) \\ &= h_0 \dots h_{\gamma^{(n-1)}(m-1)} h_{\gamma^{(n-1)}(m-1)+1} \dots h_{m-1} a^M h_0 \dots h_{s-1} \\ &= a^{M+1} h_0 \dots h_{\gamma^{(n)}(m-1)-1} \end{aligned}$$

so we can take $k_n = M + 1 + k_{n-1} = \beta(\gamma^{(n)}(m-1)) + 1 + k_{n-1}$. □

Lemma V.10. *The sequence $h_{\gamma(m-1)}, h_{\gamma(\gamma(m-1))}, h_{\gamma^{(3)}(m-1)}, \dots$ corresponds to a non-self-intersecting path*

Proof. Suppose $\prod_{i=1}^{l_1} h_{\gamma^{(i)}(m-1)} = \prod_{i=1}^{l_2} h_{\gamma^{(i)}(m-1)}$ with $l_2 > l_1$. Then, by Lemma V.9,

$$a^{k_{l_1}} h_0 \dots h_{\gamma^{(l_1)}(m-1)} = a^{k_{l_2}} h_0 \dots h_{\gamma^{(l_2)}(m-1)}$$

or,

$$e = h_{\gamma^{(l_1)}(m-1)+1} \dots h_{m-1} a^{k_{l_2} - k_{l_1} - 1} h_0 \dots h_{\gamma^{(l_2)}(m-1)}$$

Then there are two cases.

First, if $k_{l_1} = k_{l_2}$, then $\gamma^{(l_1)}(m-1) = \gamma^{(l_2)}(m-1)$ by the minimality of the expression for a. In addition, for all i with $l_1 \leq i < l_2$, $\alpha(\gamma^{(i)}(m-1)) = \emptyset$ and $\gamma^{(i)}(m-1) \neq m-1$. Therefore, for all such i,

$$\gamma^{(i+1)}(m-1) = \gamma^{(i)}(m-1) + 1 \bmod m$$

Since $\gamma^{(l_1)}(m-1) = \gamma^{(l_2)}(m-1)$, $l_2 - l_1 \geq m$. But then for some $l_1 \leq j < l_2$, $\gamma^{(j)}(m-1) = m-1$, which is a contradiction.

Second, if $k_{l_1} > k_{l_2}$, then $(k_{l_2} - k_{l_1} - 1, \gamma^{(l_2)}(m-1) + 1) \in \alpha(\gamma^{(l_1)}(m-1))$. Then $k_{l_1+1} \geq k_{l_1} + (k_{l_2} - k_{l-1} - 1) + 1 = k_{l_2}$. Since $l_2 \geq l_1 + 1$ and the sequence of k's is

non-decreasing, $l_2 = l_1 + 1$. We can then calculate $\gamma^{(l_1+1)}(m-1)$ directly. Note that

$$\begin{aligned}(\beta(\gamma^{(l_1)}(m-1)), \gamma^{(l_2)}(m-1)+1) &= (k_{l_1+1} - k_{l_1} - 1, \gamma^{(l_2)}(m-1)+1) \\ &= (k_{l_2} - k_{l_1} - 1, \gamma^{(l_2)}(m-1)+1) \\ &\in \alpha(\gamma^{(l_1)}(m-1))\end{aligned}$$

So

$$\gamma^{(l_1+1)}(m-1) = \gamma^{(l_2)}(m-1)+1 = \gamma^{(l_1+1)}(m-1)+1$$

which is again a contradiction. □

Since γ is a function from a finite set to itself, the sequence,

$$\gamma(m-1), \gamma(\gamma(m-1)), \gamma^{(3)}(m-1)\ldots$$

is eventually periodic. Let $\gamma^{(j_1)}(m-1), \ldots, \gamma^{(j_2)}(m-1)$ be a single period and take $b = \prod_{i=j_1}^{j_2} h_{\gamma^{(i)}(m-1)}$. This expression of b as a product of generators clearly fits the theorem. □

Corollary V.11. *Inescapable groups are Burnside groups*

Proof. Suppose (G, S) is a tape graph and G is not a Burnside group. Then G has an element of infinite order. By Theorem V.8, we can find some $i_0, \ldots, i_{k-1}$ such that

$$\prod_{j=0}^{N} g_{i_{j \bmod k}}$$

is distinct for each N. The sequence $\{g_{i_{j \bmod k}}\}_{j\in\mathbb{N}}$ is computable (in fact, it's regular), so G is not inescapable. □

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] T Baker, J Gill, and R Solovay, *Relativizations of the P=?NP question*, SIAM Journal on Computing **4** (1975), no. 4, 431–442.

[2] William W. Boone, *The word problem*, Proc. Nat. Acad. Sci. U.S.A. **44** (1958), 1061–1065. MR MR0101267 (21 #80)

[3] ———, *Word problems and recursively enumerable degrees of unsolvability. A first paper on Thue systems*, Ann. of Math. (2) **83** (1966), 520–571. MR MR0201499 (34 #1381)

[4] ———, *Word problems and recursively enumerable degrees of unsolvability. A sequel on finitely presented groups*, Ann. of Math. (2) **84** (1966), 49–84. MR MR0201500 (34 #1382)

[5] Irwin Fischer and Ruth R. Struik, *Nil algebras and periodic groups*, The American Mathematical Monthly **75** (1968), 611–623.

[6] E. S. Golod, *On nil-algebras and finitely approximable p-groups*, Izv. Akad. Nauk SSSR Ser. Mat. **28** (1964), 273–276. MR MR0161878 (28 #5082)

[7] E. S. Golod and I. R. Šafarevič, *On the class field tower*, Izv. Akad. Nauk SSSR Ser. Mat. **28** (1964), 261–272. MR MR0161852 (28 #5056)

[8] D. Ju. Grigor′ev, *Time complexity of multidimensional Turing machines*, Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI) **88** (1979), 47–55, 237, Studies in constructive mathematics and mathematical logic, VIII. MR MR556218 (81a:68058)

[9] Yuri Gurevich and Saharon Shelah, *Nearly linear time*, Logic at Botik, 1989, pp. 108–118.

[10] A. N. Kolmogorov and V. A. Uspenskiĭ, *On the definition of an algorithm*, Uspehi Mat. Nauk **13** (1958), no. 4(82), 3–28.

[11] Dexter Kozen, *Automata and computability*, Springer, New York, 1997.

[12] ———, *Theory of computation*, Springer, New York, 2006.

[13] Ashish V. Naik, Kenneth W. Regan, and D. Sivakumar, *On quasilinear-time complexity theory*, Theoretical Computer Science **148** (1995), no. 2, 325 – 349, Selected Papers of the Eleventh Symposium on Theoretical Aspects of Computer Science.

[14] P. S. Novikov, *On algorithmic unsolvability of the problem of identity*, Dokl. Akad. Nauk. SSSR **85** (1952), 709–719.

[15] Giuseppe Pirillo, *Repetitive mappings and morphisms*, Combinatorics on Words, by M. Lothaire, Addison-Wesley Publishing Company, 1983.

[16] Mark V. Sapir, Jean-Camille Birget, and Eliyahu Rips, *Isoperimetric and isodiametric functions of groups*, The Annals of Mathematics **156** (2002), no. 2, pp. 345–466 (English).

[17] C. P. Schnorr, *Satisfiability is quasilinear complete in nql*, J. ACM **25** (1978), 136–145.

[18] A. Schönhage, *Universelle Turing speicherung*, Automatentheorie und Formale Sprachen (Hotz Dörr, ed.), Bibliogr. Institut, Mannheim, 1970, pp. 369–383.

[19] John C. Stillwell, *Efficient computation in groups and simplicial complexes*, Transactions of the American Mathematical Society **276** (1983), no. 2, pp. 715–727 (English).

[20] Alan M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society **42** (1936), 230–265.