APPROXIMATING THE RANK OF A HOMOMORPHISM
USING A PROLOG BASED SYSTEM
by
Richard Kenneth Little
B.Sc., The University of Northern British Columbia, 1996

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ARTS
in
INTERDISCIPLINARY STUDIES

© Richard Kenneth Little, 2000
THE UNIVERSITY OF NORTHERN BRITISH COLUMBIA
May 2000

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
Abstract

A system of Prolog based programs for the purpose of approximating the rank of algebraic operations of finite unary algebras is presented. The rank function is a measure of finite algebras and their algebraic operations. Rank is a recursive function used in universal algebra and was first introduced as a tool for proving strong dualizability. Logic programming, particularly Prolog, is commonly used in natural language processing, an area of study devoted to the use of computers to understand human (natural) languages.

One goal of this thesis is to explore a relationship between the fields of Mathematics and Computer Science through the application of logic programming techniques on structures from universal algebra. This thesis is motivated by the idea that when universal algebra is viewed as a language, the ideas of natural language processing can be used to create a computer system which approximates rank. The outcome of the research is a computational model that computes the $Kth$ approximation of rank. A set of Prolog programs that act as useful tools on algebraic structures are created.
Acknowledgments

I would like to thank, first of all, my co-supervisors, Jennifer Hyndman and Charles Brown. Jennifer for introducing me to the world of universal algebra and the countless hours she spent teaching me how to understand rank. Charles for introducing me to the world of Prolog and the countless hours he spent teaching me how to understand coding rank. I also wish to express my gratitude to Professor Bruno Zumbo for breathing new life into my waning enthusiasm for the research. Thank you Dr. David Casperson for putting the time and effort that you did into reading the thesis. Furthermore, I thank the rest of the Mathematics and Computer Science department at UNBC for supporting and guiding me since my first day here.

Thank you my fellow grad students, Erin Beveridge and David Kubert, for listening and helping throughout. Thank you Roberta Bilous, for putting up with my endless requests with a smile on your face. Thank you to all the roommates (909, 1221, 1282, 1722, and 821) and friends (the university people and the ball people) I have acquired during my Prince George stay, its been real. Thanks Brian Bowman for showing me I can write. Thanks Robert Neil Diaz, Esq. for dragging my lazy butt out here and always being there for me.

A special thanks to my parents, David and Flora Little, and my brother, Christopher Little, for always supporting my decisions in life without question. Finally, I would like to thank the most wonderful woman in the world, who is also the love of my life, Tracy Bulman. Thanks babe.
Contents

Abstract ii

Acknowledgments iii

Contents iv

List of Tables vi

List of Figures viii

0 Introduction 1

1 Universal Algebra and Rank 3

1.0 Introduction 3

1.1 Universal Algebra 4

1.2 Rank 8

1.3 Summary 16

2 Computational Model 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0 Introduction</td>
<td>17</td>
</tr>
<tr>
<td>2.1 Problem Analysis - The Experimental Design</td>
<td>18</td>
</tr>
<tr>
<td>2.2 Logic Programming and Prolog</td>
<td>21</td>
</tr>
<tr>
<td>2.3 The Computational Model</td>
<td>26</td>
</tr>
<tr>
<td>2.4 Complexity Analysis</td>
<td>43</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>50</td>
</tr>
<tr>
<td>3 Results and Discussion</td>
<td>51</td>
</tr>
<tr>
<td>3.0 Introduction</td>
<td>51</td>
</tr>
<tr>
<td>3.1 make_projections/4</td>
<td>52</td>
</tr>
<tr>
<td>3.2 rank/6</td>
<td>65</td>
</tr>
<tr>
<td>3.3 Summary</td>
<td>82</td>
</tr>
<tr>
<td>4 Conclusion</td>
<td>83</td>
</tr>
<tr>
<td>4.0 Introduction</td>
<td>83</td>
</tr>
<tr>
<td>4.1 Conclusion</td>
<td>83</td>
</tr>
<tr>
<td>4.2 Future Work</td>
<td>85</td>
</tr>
<tr>
<td>4.3 Summary</td>
<td>87</td>
</tr>
</tbody>
</table>

Bibliography                                  87

A Appendix: Tables                            91

B Appendix: Code                              119
List of Tables

2.1 Big-Oh analysis on selected predicates. ........................................... 45

3.1 Running times and complexity of make_projections/4 on algebras M_2
    and P_2. Size m = 1. Only 1 subalgebra for each k. .......................... 54

3.2 Times to run make_projections/4 on algebras of size m = 2, for 1 \leq k \leq 3. 57

3.3 Times to run make_projections/4 on algebras of size m = 2, for 1 \leq k \leq 4. 58

3.4 Times to run make_projections/4 on algebras of size m = 3, for 1 \leq k \leq 2. 60

3.5 Times to run make_projections/4 on algebra, M_12, of size, m = 3, for
    1 \leq k \leq 3. ................................................................................. 61

3.6 Times to run make_projections/4 on algebras of size, m = 4, for 1 \leq
    k \leq 2. ......................................................................................... 62

3.7 Average run times over 1 \leq m \leq 4, and 1 \leq k \leq 4. ......................... 63

3.8 The results of running rank/6 on the algebraic operations from algebra,
    B_{15}, to algebra, M_1. ................................................................. 68

3.9 The results of running rank/6 on the algebraic operations from algebra,
    B_{16}, to algebra, M_1. ................................................................. 69
3.10 The results of running rank/6 on the algebraic operations from algebra, 
\( B_{91} \), to algebra, \( M_9 \). .................................................. 71

3.11 The results of running rank/6 on the algebraic operations from algebra, 
\( B_{91} \), to algebra, \( M_9 \) (continued). ..................................... 72

3.12 The results of running rank/6 on the algebraic operations from algebras, 
\( B_{11} \) and \( B_{12} \), to algebra, \( P_1 \). ........................................ 76

3.13 The results of running rank/6 on the algebraic operations from algebras, 
\( B_{121} \) and \( B_{122} \), to algebra, \( P_{12} \). ....................................... 77

3.14 The results of running rank/6 on the algebraic operations from algebras, 
\( B_{41} \) and \( B_{42} \), to algebra, \( P_4 \). .......................................... 78

3.15 The results of running rank/6 on the algebraic operations from algebras, 
\( B_{111} \), \( B_{112} \) and \( B_{113} \), to algebra, \( P_{11} \). ............................ 80

3.16 The results of running rank/6 on the algebraic operations from algebras, 
\( B_{101} \), \( B_{102} \) and \( B_{103} \), to algebra, \( P_{10} \). ............................ 81
List of Figures

1-1 (a) An algebra \( M_1 \); (b) a subalgebra \( B_1 \) of \((M_1)^2\); (c) \((B_1)'\) a subalgebra of \((M_1)^3\); and (d) the algebra \((M_1)^2\) ........................................... 10

1-2 The commuting diagram for lifting. ........................................... 10

1-3 The commuting diagram for rank. ........................................... 11

1-4 (a) The algebra \( M_{1+k}/Y \), (b) a subalgebra \( B \) of \( M_1 \), and (c) the algebra \( B' \) a subalgebra of \( M_{1+k}^{1+k} \) ........................................... 12

1-5 (a) An algebra \( M \) and (b) \( C \) a subalgebra of \( M^2 \) ........ 14

1-6 (a) The algebra \( M \) and (b) subalgebra \( B \). ........................................... 16

1-7 Algebra \( P_1 \). ........................................... 16

2-1 Algebras \( M_1 \) and \( P_1 \). ........................................... 27

2-2 Algebras \( M_3 \) and \((M_3)^2\). ........................................... 28

2-3 Algebras \( M_9 \) and \( M_{12} \). ........................................... 49

3-1 Algebras \( M_2 \) and \( P_2 \). ........................................... 53

3-2 Algebras \( M_3, M_4, \) and \( M_5 \). ........................................... 56
3-3 Algebras $P_3$, $P_4$, $P_5$, and $P_6$ ........................................ 56
3-4 Algebras $P_7$, $P_8$, and $P_9$. ........................................ 57
3-5 Algebras $M_6$, $M_7$, and $M_8$. ........................................ 59
3-6 Algebras $M_9$, $M_{10}$, and $M_{11}$. ................................. 59
3-7 Algebras $M_{12}$, $P_{10}$, $P_{11}$, and $P_1$. ......................... 60
3-8 Algebras $M_1$, and $P_{12}$. ........................................ 62
3-9 (a) The algebra $M_1$ and (b) the algebra $P_1$ ....................... 67
3-10 The algebra, $B_{12}$, a subalgebra of $(P_1)^2$. .................... 75
Chapter 0

Introduction

This thesis presents a system of Prolog based programs for the purpose of approximating the rank of homomorphisms of finite unary algebras. The rank function, introduced by Ross Willard in 1998 [22], is a measure of finite algebras and their operations. Rank is a recursive function of universal algebra and was first introduced as a tool for proving strong dualizability. Logic programming, particularly Prolog, is commonly used in natural language processing, an area of study devoted to the use of computers to understand human (natural) languages [8].

The general goal of this thesis is to explore a relationship between the fields of Mathematics, Computer Science and Natural Language Processing. This thesis presents the hypothesis that when universal algebra is viewed as a language, the ideas of natural language processing can be used to create a computer system which approximates rank. The method used to make this link is the design and implementation of Prolog programs
which treat universal algebra as a language. The primary purpose of these programs is to create the tools necessary to approximate rank. This link is established through the process of writing code and demonstrated with the results obtained from trial runs of the computational model. A further goal of this thesis is to gain a better understanding of the nature of the rank function.

The programs contained in this thesis approximate the rank of algebraic operations of finite unary algebras using a Prolog based system. In Chapter 1, the necessary definitions and theorems from universal algebra are presented as background for the definition of rank. Additionally, Chapter 1 introduces the definition of the rank function, and examines results and examples currently known in this area. Chapter 2 presents the computational model as well as a discussion of the merits of using Prolog for this purpose. Chapter 2 also contains an in-depth look at specific pieces of the code, and their complexity analysis. Chapter 3 includes a discussion of the results obtained from the simulations and a comparison of the actual runtimes to the complexity analysis of Chapter 2. Finally, Chapter 4 contains the conclusions of the thesis and puts forth a discussion of possible future work and research in this area. The thesis also contains an appendix of the results presented in tables and an appendix of the code with detailed comments about the various files and functors used.
Chapter 1

Universal Algebra and Rank

1.0 Introduction

For a greater understanding of the computational model of Chapter 2, an in-depth look at the rank function is necessary. The rank function is a tool used for measurement of finite algebras and algebraic operations in the field of universal algebra. Rank was first introduced as a means of proving strong dualizability in the area of duality theory. Although this thesis does not deal with duality issues, it is worthwhile to point out the original motivation of the rank function. The dualizability of an algebra is of interest but is beyond the scope of this study. This thesis is a study of the mechanisms of the rank function and results obtained with it. As such, this chapter presents the original definition of rank as well as current results in universal algebra based on the rank function.

The first section of Chapter 1 is a brief overview of the definitions and theorems from
universal algebra used in calculating rank. The second section defines rank, the $Kth$ approximation of rank, and presents some recent results in the area.

### 1.1 Universal Algebra

The concepts from universal algebra that are required include those of subalgebra, congruence, product, and homomorphism. These are developed below, restricted to unary algebras, closely following the notation and terminology of [3].

For $A$, a nonempty set, and $n$, a nonnegative integer, we define $A^0 = \{\emptyset\}$, and, for $n > 0$, $A^n$ is the set of $n$-tuples of elements from $A$. The cardinality of a set, $A$, written $|A|$, is the number of elements in the set. An $n$-ary relation on a set, $A$, is a subset of $A^n$. If $n = 2$, it is called a binary relation on $A$. A binary relation $\theta$ on $A$ is an equivalence relation on $A$ if, for any $a$, $b$, and $c$ from $A$, it satisfies:

\[
\begin{align*}
E1: \quad & (a, a) \in \theta \\
E2: \quad & (a, b) \in \theta \implies (b, a) \in \theta \\
E3: \quad & (a, b) \in \theta \text{ and } (b, c) \in \theta \implies (a, c) \in \theta
\end{align*}
\]

(reflexivity); (symmetry); (transitivity).

The function, $f : A \rightarrow B$, is said to be one-to-one if $f(a_1) = f(a_2)$ implies $a_1 = a_2$. Furthermore, the function, $f : A \rightarrow B$, is onto if for every $b \in B$ there is an $a \in A$ with $f(a) = b$. An $n$-ary operation (or function) on $A$ is any function, $f$, from $A^n$ to $A$; $n$ is the arity of $f$. A finitary operation is an $n$-ary operation, for some finite $n$. The image of $(a_1, \ldots, a_n)$ under an $n$-ary operation, $f$, is denoted by $f(a_1, \ldots, a_n)$. An operation is
unary if its arity is 1.

A language (or type) of algebras is a set $\mathcal{F}$ of function symbols such that a nonnegative integer $n$ is assigned to each member $f$ of $\mathcal{F}$. This integer is called the arity of $f$, and $f$ is said to be an $n$-ary function symbol. The subset of unary function symbols in $\mathcal{F}$ is denoted $\mathcal{F}_1$.

An algebra $A$ of type $\mathcal{F}_1$ (a unary algebra) is an ordered pair, $A = \langle A, F \rangle$, where $A$ is a nonempty set and $F$ is a finite family of unary operations on $A$ indexed by the language, $\mathcal{F}_1$, such that corresponding to each unary function symbol, $f$, in $\mathcal{F}_1$ there is a unary operation, $f^A$, on $A$. The set, $A$, is called the universe (or underlying set) of $A$, and the $f^A$'s are called the fundamental operations of $A$. In practice it is common to write just $f$ for $f^A$ and often to write $\langle A, f_1, \ldots, f_k \rangle$ for $\langle A, F \rangle$.

A mono-unary algebra, $M = \langle M, f \rangle$, is an algebra with one unary function, a bi-unary algebra, $P = \langle P, f_1, f_2 \rangle$, is an algebra with two unary functions, and so on. Throughout the rest of the thesis an algebra refers to a unary algebra.

Let $A$ and $B$ be two algebras. Then $B$ is a subalgebra of $A$ if $B \subseteq A$ and every fundamental operation of $B$ is the restriction of the corresponding operation of $A$, i.e., for each function symbol $f$, $f^B$ is $f^A$ restricted to $B$; we write simply $B \leq A$. A subuniverse of $A$ is a subset $B$ of $A$ which is closed under the fundamental operations of $A$, i.e., if $f$ is a fundamental operation of $A$ and $a \in B$ we require $f (a) \in B$. Thus if $B$ is a subalgebra of $A$, then $B$ is a subuniverse of $A$. Note that the empty set may be a subuniverse, but it is not the underlying set of any subalgebra.
Let \( A \) be an algebra and \( X \) a subset of \( A \), then

\[
Sg(X) = \cap \{ B : X \subseteq B \text{ and } B \text{ is a subuniverse of } A \}.
\]

The notation, \( Sg(X) \), is read as the subuniverse generated by \( X \).

**Theorem 1** Let \( A = (A,F) \) be an algebra and \( X \) a subset of \( A \). Then, \( Sg(X) \) is generated recursively, as follows. \( Sg(X) = \cup X_{i \in I} \), where \( X_0 = X \) and \( X_{n+1} = X_n \cup \{ f(a) : f \in F, a \in X_n \} \).

Suppose \( A \) and \( B \) are two algebras. A function, \( h : A \rightarrow B \), is called a homomorphism from \( A \) to \( B \) if

\[
h(f^A(a)) = f^B(h(a))
\]

for all fundamental operations, \( f \), in \( A \) and each \( a \) from \( A \). A homomorphism, \( h : A \rightarrow B \), is an embedding of \( A \) into \( B \) if \( h \) is one-to-one and it is an isomorphism from \( A \) to \( B \) if it is one-to-one and onto.

Let \( A \) be an algebra and \( B \leq A^n \). The homomorphisms, \( h : B \rightarrow A \), for all positive integers, \( n \), are called the algebraic operations of \( A \).

**Theorem 2** Suppose \( h : A \rightarrow B \) and \( g : B \rightarrow C \) are homomorphisms. Then the composition \( g \circ h \) is a homomorphism from \( A \) to \( C \).

**Theorem 3** If \( \alpha : A \rightarrow B \) is an embedding, then \( \alpha(A) \) is a subuniverse of \( B \).
If $\alpha : \mathbf{A} \rightarrow \mathbf{B}$ is an embedding, $\alpha (\mathbf{A})$ denotes the subalgebra of $\mathbf{B}$ with universe $\alpha (\mathbf{A})$.

Let $\mathbf{A}$ be an algebra and let $\theta$ be an equivalence relation. Then $\theta$ is a congruence on $\mathbf{A}$ if for each function symbol $f \in \mathcal{F}_1$ and elements $a, b \in \mathbf{A}$, if $(a, b) \in \theta$ then

$$\langle f^\mathbf{A} (a), f^\mathbf{A} (b) \rangle \in \theta.$$ 

Let $h : \mathbf{A} \rightarrow \mathbf{B}$ be a homomorphism. Then the kernel of $h$, written $\ker (h)$, is defined by

$$\ker (h) = \{ (a, b) \in \mathbf{A}^2 : h (a) = h (b) \}.$$ 

**Theorem 4** Let $h : \mathbf{A} \rightarrow \mathbf{B}$ be a homomorphism. Then $\ker (h)$ is a congruence on $\mathbf{A}$.

Let $\mathbf{A}$ be an algebra and let $\theta$ be a congruence on $\mathbf{A}$. The natural map $\kappa : \mathbf{A} \rightarrow \mathbf{A}/\theta$ is defined by $\kappa (a) = a/\theta$.

**Theorem 5** The natural map from an algebra to a quotient of the algebra is an onto homomorphism.

Let $\mathbf{A}_1, \ldots, \mathbf{A}_n$ be $n$ algebras. Define the product, $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$, to be the algebra whose universe is the set, $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$, and such that for $f \in \mathcal{F}_1$, $a_1 \in \mathbf{A}_1, \ldots, a_n \in \mathbf{A}_n$,

$$f^{\mathbf{A}_1 \times \cdots \times \mathbf{A}_n} ((a_1, \ldots, a_n)) = \langle f^{\mathbf{A}_1} (a_1), \ldots, f^{\mathbf{A}_n} (a_n) \rangle.$$
The mapping

$$\pi_i : A_1 \times \cdots \times A_n \rightarrow A_i, \quad i \in \{1, \ldots , n\},$$

defined by

$$\pi_i ((a_1, \ldots , a_n)) = a_i,$$

is called the projection map on the $i^{th}$ coordinate of $A_1 \times \cdots \times A_n$.

**Theorem 6** For $i = 1, \ldots , n$, the mapping, $\pi_i : A_1 \times \cdots \times A_n \rightarrow A_i$, is a surjective homomorphism from $A = A_1 \times \cdots \times A_n$ to $A_i$.

If $A_i = A$ for all $i \in \{1, \ldots , n\}$, then the product is written $A^n$ and is called it a power of $A$.

### 1.2 Rank

This section provides us with the definition of rank. It also gives some of the current results which relate to rank. The definition of rank is taken directly from [10], which is based on the original definition given by Ross Willard in [22]. In Willard’s paper, he developed the concept of rank and then used it to prove that all finite algebras with a finite rank that are dualizable are strongly dualizable [22, Theorem 4.1]. This, in turn, was motivated by the attempt to prove that the ring, $\mathbb{Z}_4$, is strongly dualizable in [6], in which it is proved implicitly that the rank of any finite commutative ring with identity whose Jacobson radical $J$ satisfies $J^2 = 0$ is less than or equal to one. To reiterate a
point made earlier, duality theory is not covered by this thesis, but some terms and ideas are mentioned to provide the motivation for rank.

Let $M$ be a fixed finite algebra, $n$ a positive integer, and $B$ a subalgebra of $M^n$. Also, let $h$ be a member of the set of all homomorphisms from $B$ to $M$ (written $Hom(B, M)$). For algebra, $B'$, a subalgebra of $M^{n+k}$, for some finite $k$, the notation, $B \equiv >_{\sigma} B'$, denotes $\sigma : B \rightarrow B'$ is an embedding via repetition of some coordinates. The homomorphism, $h' = \sigma^{-1} \circ h$, is the natural extension of $h$ to $B'$. Throughout the thesis, in the diagrams used to represent algebras, an arrow from element, $x$, to element, $y$, denotes the fact that $f(x) = y$. For notational convenience, $(x_1, x_2, \ldots, x_n) \in M^n$ is denoted by $x_1x_2\cdots x_n$.

**Example 1** Referring to Figure 1-1, let $M_1$ be the algebra represented by Figure 1-1(a), which has universe $M = \{0, a, b, c\}$. Figure 1-1(b) shows an example of a subalgebra $B_1$ of $(M_1)^2$. Letting $k = 1$, algebra, $(B_1)'$, is a subalgebra of $(M_1)^3$, where $B_1$ embeds into $(B_1)'$ by $\sigma(ba) = bba$, a repetition of the first coordinate. Algebra, $(B_1)'$, appears in Figure 1-1(c).

Let $B' \leq C \leq D \leq M^{n+k}$ and let $Y \subseteq Hom(D, M)$. The algebra, $D/Y$, denotes the algebra $D/\cap \{\ker(g) : g \in Y\}$ and $C/Y$ the algebra $C/\cap \{\ker(g|_C) : g \in Y\}$. The set, $Y$, separates $B'$ if $\cap \{\ker(g|_{B'}) : g \in Y\} = 0_{B'}$. The notation, $0_{B'}$, represents the congruence, $0_{B'} = \{(x, x) : x \in B'\}$. The homomorphism, $h'$, lifts to $C/Y$ if $Y$ separates $B'$ and there exists a map $\mu$ such that the diagram in Figure 1-2 commutes. A diagram is said to **commute** if the composition of functions over any path of arrows from the same starting algebra to the same final algebra results in the same homomorphism.
Figure 1-1: (a) An algebra $M_1$; (b) a subalgebra $B_1$ of $(M_1)^2$; (c) $(B_1)'$ a subalgebra of $(M_1)^3$; and (d) the algebra $(M_1)^2$.

Definition 7 (Rank) Given a homomorphism, $h : B \to M$, then $\text{rank}(h) \leq 0$ if and only if $h$ is a projection. The $\text{rank}(h) \leq \alpha$ if and only if there exists a finite $N$ such that for all nonnegative integers, $k$, for all subalgebras, $D$, of $M^{n+k}$ and for all commuting diagrams, shown in Figure 1-3, where the homomorphism $h'$ lifts via $h^+$ to $D$, there exists $Y \subseteq \text{Hom}(D, M)$ such that $|Y| \leq N$, $h'$ lifts to $C/Y$, and $\text{rank}(g|_C) < \alpha$ for all $g \in Y$. 

Figure 1-2: The commuting diagram for lifting.
The equality, \( \text{rank} (h) = \alpha \), holds if and only if \( \text{rank} (h) \leq \alpha \) and not \( \text{rank} (h) < \alpha \). If there is no such finite \( \alpha \), then write \( \text{rank} (h) = \infty \). The rank of \( M \), \( \text{rank} (M) \), is the supremum of the ranks of all the algebraic operations of \( M \). The following example gives an illustration of the steps involved in finding the rank of a homomorphism.

**Example 2** Let \( M_1 \) be the algebra given in Example 1 on page 9. Let \( B \) be the two element subalgebra of \( M_1 \), \( \{0, b\} \) of Figure 1-4(b) and let \( n = 1 \). Also, let \( N \geq 1 \) and \( h : B \to M_1 \) be the homomorphism given by \( h(b) = a \) and \( h(0) = 0 \). Let \( \bar{x} = (x, x, \ldots, x) \in (M_1)^{1+k} \), where \( x \in M_1 \). For any fixed nonnegative integer, \( k \), the algebra, \( B' \), in Figure 1-4(c), is a subalgebra of \( (M_1)^{1+k} \) and \( B \) embeds into \( B' \) by repetition of coordinates. The homomorphism, \( h' : B' \to M_1 \), is given by \( h'(\bar{b}) = a \) and \( h'(0) = 0 \). For all \( B \Rightarrow C \leq D \leq M_1^{1+k} \), \( h' \) lifts to \( D \) via the homomorphism, \( h^* : D \to M_1 \), defined by \( h^*(\bar{b}) = a \), \( h^*(0) = 0 \), and sending everything else to 0.

The set, \( Y = \{ \pi_i \} \subseteq \text{Hom} (D, M) \), is a set of one homomorphism from \( D \) to \( M_1 \) that separates \( B' \). The algebra, \( M_1^{1+k}/Y \), is the factor algebra \( \{ [\bar{0}]_Y, [\bar{a}]_Y, [\bar{b}]_Y, [\bar{c}]_Y \} \) of Figure 1-4(a). There exists a homomorphism, \( \mu : C/Y \to M_1 \), defined by \( \mu([\bar{b}]_Y) = a \), \( \mu([\bar{a}]_Y) = b \), and \( \mu([\bar{c}]_Y) = c \).
\( \mu([0]_Y) = 0 \) and \( \mu \) sends everything else to 0. The homomorphism, \( h' \), lifts to \( C/Y \) by \( \mu \). Since \( |Y| = 1 \leq N \) and \( \text{rank}(\pi_1) \leq 0 \), then \( \text{rank}(h) \leq 1 \).

\[
\begin{aligned}
(a) \quad M^{1+k} & \quad \begin{array}{c}
\bullet \quad [\hat{a}]_Y \\
\downarrow \quad [\hat{b}]_Y \\
\downarrow \quad [0]_Y
\end{array} \\
(b) \quad B & \quad \begin{array}{c}
\bullet \quad b \\
\downarrow \quad 0
\end{array} \\
(c) \quad B' & \quad \begin{array}{c}
\bullet \quad \hat{b} \\
\downarrow \quad \hat{0}
\end{array}
\end{aligned}
\]

Figure 1-4: (a) The algebra \( M^{1+k}_1/Y \), (b) a subalgebra \( B \) of \( M_1 \), and (c) the algebra \( B' \) a subalgebra of \( M^{1+k}_1 \).

The definition of rank includes the universal quantification ‘for all nonnegative integers \( k \).’ The following definition changes this universal quantification to a bounded quantification.

**Definition 8** The \( K \)-th approximation of rank is the same as rank (Definition 7), except that ‘for all nonnegative integers, \( k \),’ is replaced by ‘for all \( k \leq K \) for a fixed finite \( K \).’

The following results are taken directly from [10], in which it is proven that mono- unary algebras have rank at most two and thus are strongly dualizable. It is necessary to start with some preliminary definitions, lemmas and theorems before getting to the main point of the paper.

A **connected component** of a mono-unary algebra is a subalgebra \( B \) that is maximal with respect to the property that for all \( a, b \in B \), \( f^m(a) = f^s(b) \) for some \( m, s \geq 0 \).

Let \( M = (M, f) \) be a finite mono-unary algebra. Let \( C \leq M^n \), where \( n \) is a finite positive integer. Let \( A \) be a connected component of \( C \). The **essential components**
of \( A \) are the minimum set of connected components of \( M \) that contain \( \pi_i(A) \) for all projections \( \pi_i \).

The core of a connected mono-unary algebra is a nonempty subalgebra on which \( f \) is a one-to-one function. In a finite, connected mono-unary algebra the core will be of the form \( \{ a, f(a), f^2(a), \ldots, f^{k-1}(a) \} \), where \( f^k(a) = a \) and \( k \) an integer. An arbitrary finite mono-unary algebra may have several cores.

The circumference of the connected component \( A \), \( \text{circ} (A) \), is the least \( k \) such that for some \( a \in A \), \( f^k(a) = a \). That is, the circumference of a finite, connected mono-unary algebra is the size of the core.

For \( x \in C \) such that \( x \) is not in a core but \( f(x) \) is in a core, the set

\[
\{ y \in B | f^m(y) = x \text{ for some } m \geq 0 \}
\]

is a branch. For \( x \in C \), \( x \) is a branch element if \( x \) is an element of a branch. For \( x \) a branch element, the coheight of \( x \) is the greatest \( k \) such that there exists a \( y \) with \( f^k(y) = x \).

For \( x \) a core element \( \text{coheight} (x) = \infty \). An illustration of the above definitions is made in the following example.

**Example 3** Let \( M \) be the mono-unary algebra \( \{ a, b, 0, 1 \} \) shown in Figure 1-5(a). \( M \) has two connected components, call them \( M_a = \{ a, 0, 1 \} \) and \( M_\beta = \{ b \} \). \( C = \{ aa, ab, 0b, 00, 1b, 11 \} \), given in Figure 1-5(b), is a subalgebra of \( M^2 \). \( C \) also has two connected components, call them \( A_\alpha = \{ aa, 00, 11 \} \) and \( A_\beta = \{ ab, 0b, 1b \} \). For \( A_\alpha \), the
essential component is $M_\alpha$, it has the core $\{00, 11\}$, $\text{circ}(A_1) = 2$ and it contains one branch with one branch element $aa$ with coheight $(aa) = 0$. The connected component, $A_\beta$, has essential components $M_\alpha$ and $M_\beta$, core $\{0b, 1b\}$, circumference 2 and branch element $ab$ with coheight $(ab) = 0$.

![Figure 1-5: (a) An algebra $M$ and (b) $C$ a subalgebra of $M^2$.](image)

**Lemma 9** Let $A$ be a connected component of $C$ and $M_\alpha$ an essential component of $A$. Then the circumference of $A$ is a (positive) integer multiple of the circumference of $M_\alpha$.

For $A$ a connected component in $C$ and $M_\alpha$ an essential component of $A$, to wrap $A$ around $M_\alpha$ means to define a homomorphism, $q$, from $A$ to the core of $M_\alpha$ recursively as follows. Pick $a \in A$ and $d$ in the core, $L$, of $M_\alpha$. Let $q(a) = d$ and $q(f^m(a)) = f^m(d)$. If $q$ is defined on $f(x)$ but not defined on $x$ then let $q(x) \in f^{-1}(q(f(x))) \cap L$, which has exactly one element in it, i.e. the one $l \in L$ that is sent by $f$ to $q(f(x))$.

**Lemma 10** Let $B_1, \ldots, B_i$ be the disjoint connected components of an algebra $B$. Let $h_i : B_i \rightarrow M$ be homomorphisms. Then $h = \cup h_i$ is a homomorphism from $B$ to $M$.

For a branch element $b = (b_1, \ldots, b_n) \in B$ define homomorphism, $g_b : B \rightarrow M$, as follows. Let $A_i$ be the connected component of $B$ with $b \in A_i$. Pick $u$ in an essential
component, $M_\alpha$, of $A_1$ such that $\coheight_{M}(u)$ is finite and maximal over all elements in essential components of $A_1$. Let $t = \coheight_{B}(b)$. Define $g_b(b) = u$. For any $x$ in $B$ and any $s$ with $1 \leq s \leq t$ where $f^s(x) = b$ define $g_b(x) = f^{t-s}(u)$. Wrap the remaining elements of $A_1$ around the core of $M_\alpha$ specifying $g_b(f(b)) = f(u)$. By Lemma 10, we may extend $g_b$ by wrapping each connected component $A_2$, distinct from $A_1$, around an essential component of $A_2$.

**Theorem 11** For $b$ a branch element of $B$, $\text{rank } (g_b) \leq 1$.

**Theorem 12** For $B \leq M''$ and $h : B \to M$ a homomorphism, $\text{rank } (h) \leq 2$.

**Example 4** Consider the algebra $M_1$, with four elements $\{0, a, b, c\}$, of Figure 1-6(a). Here we illustrate that $M_1$ has rank 2. Let $B = \{0, a\}$, the universe of the algebra, $B$, shown in Figure 1-6(b). Consider the homomorphism $\tilde{h} : B \to M_1$ given by $h(a) = b$ and $h(0) = 0$. Fix $N \geq 1$ and let $k \geq N$. Let $D = M_1^{k+1} \setminus \{c\}$ and let $B' = aB' \leq C \leq D$. Then $h'$, the natural extension of $h$ to $B'$, lifts to $D$. Neither $\tilde{a}$ has a pre-image in $D$ nor does $h'(\tilde{a})$ have a pre-image in $M_1$. Let $Y$ be a collection of fewer than $N$ projections, then $[\tilde{a}]_Y$ has a pre-image in $D/Y$. Thus $h'$ does not lift to $D/Y$. In order for $h'$ to lift to $D/Y$, $Y$ must have either $k + 1$ projections or a non-projection. Since $k$ can be arbitrarily large and $|Y| \leq N$ which is fixed, $Y$ must contain more than just projections and thus by Theorem 11 $\text{rank } (h) \geq 2$. By Theorem 12, $\text{rank } (h) = 2$.

Finally, in [11] Hyndman and Willard show an example of a finite algebra, $P_1$, which is dualizable but not fully dualizable. This algebra is shown in Figure 1-7. This result
Figure 1-6: (a) The algebra $M$ and (b) subalgebra $B$.

implies that the rank of $P_1$ is infinite by [22, Theorem 4.1].

![Diagram of $P_1$]

Figure 1-7: Algebra $P_1$.

1.3 Summary

The rank of an algebra and its algebraic operations is based on the terms of universal algebra given in this chapter. The rank of an algebra is a tool used for proving strong dualizability. It has been shown that finite algebras with a finite rank that are dualizable are strongly dualizable, mono-nary algebras have a finite rank, that the ring $\mathbb{Z}_4$ has rank one, and that the rank of $P_1$ is infinite.
Chapter 2

Computational Model

2.0 Introduction

This chapter provides a brief overview of the goals of creating the computational model and a breakdown of the necessary steps taken to achieve these goals. The broader goal of this research is to unify the application of the fields of Mathematics, Computer Science and Formal Languages in a relevant and practical way. The specific goal is to create a computer based system for the purpose of approximating the rank of a finite unary algebra or algebraic operations of a finite unary algebra using the methods of language models. To attain the specific goal of approximating rank computationally it is necessary to break it down into several sub-goals. These sub-goals include the design of the model, selecting a language to work with, writing and implementing code, and analyzing the code. The goal of using Mathematics, Computer Science, and Formal Languages in unison is attained by
implementing the code, running trials, and gathering results. When the results provide support to known theoretical results and are consistent with what is known then we are confident that the application of these fields is correctly united.

The first section of Chapter 2 looks at the design of the computational model. The second section points out the advantages of using logic programming, specifically Prolog, for this thesis. The third section in this chapter presents the computational model. This section is where the ideas of the previous two sections are brought together. In the final section a detailed complexity analysis of the code is given. The complexity analysis consists of timing analysis done on the algorithms used, in Big-Oh notation.

2.1 Problem Analysis - The Experimental Design

One of the goals of this research is to create a computational model using the ideas of language processing to approximate rank. In order to write programs for any purpose it is necessary for a programmer to decide on a programming language. Furthermore, before programmers select a language they need to outline the problem and define what the programming language needs to provide. This section is the starting block for the programming process, in which the problem at hand is dissected and studied to provide motivation for the choice of programming language.

The purpose of the rank function is to measure algebras and homomorphisms. The first thing needed is to define data structures to represent algebras and homomorphisms on algebras. An algebra is defined in Chapter 1 as a set of elements, or universe, with
a list of unary functions defined on the set. A homomorphism is a mapping, defined on
the elements of an algebra, from the algebra to another, which respects the operations.
There is no restriction on the elements that can be used in an algebra or a homomor-
phism. For both, the best choice of data structure allows for manipulation of general
types of elements. Representing the elements symbolically, allows for more freedom in the
choice of algebras and homomorphisms than when confined to numerical representation.
Furthermore, it would be beneficial to have a readable representation of an algebra and
homomorphism. Although the data structures corresponding to algebra and homomor-
phism are similar in nature, it is necessary to be able to easily distinguish between the
two.

The rank of an algebraic operation is recursively defined using ranks of simpler alge-
braic operations. The rank of an algebra is defined in terms of the ranks of the algebraic
operations of that algebra. This implies that it is necessary to have a store, or database,
of homomorphisms of known rank. To start, the database would consist of projections
which by definition are rank less than or equal to zero. It is highly probable that this
database will be very large and will constantly grow. Optimistically, the database should
have the capability of storing large amounts of information while at the same time be eas-
ily and quickly accessible to the calling program. In addition, it would be advantageous
to be able to add new homomorphisms to the database as their rank is discovered.

The definition of approximate rank is recursive with base case determined by projec-
tions. Moreover, the procedures used to generate subalgebras and homomorphisms from
a given set are recursive. As a result, it is most effective to write recursive computer programs to compute approximate rank.

Generating all the possible subalgebras of a given algebra is necessary in creating the database of projections and in approximating rank. Furthermore, in the definition of rank it may be necessary to exhaust all the possible sets of homomorphisms with known rank. Therefore, the model should be capable of generating and exhausting all the possible solutions to a particular problem. In some cases, it is necessary to use all the solutions possible in the search, while at other times one need only search the possible outcomes until an appropriate outcome is reached. The difference here depends on the quantification, it reflects the difference between the statements for all and there exists. These restrictions imply a need to have strict control of the overall flow of the program at any given time. To summarize, the computational model must be capable of exhausting all possibilities for a single goal while at the same time knowing when to cut that search off.

So far the discussion has centered around the general issues and encompassing feel of programming rank. One more issue needs to be addressed. To properly code the approximation of rank, many tools from universal algebra are used and need to be written as sub-routines to be called by the rank routine. These include, embedding an algebra, B, into algebra, B', by the repetition of some k coordinates; separating an algebra, B', by a set of homomorphisms, Y; factoring the algebra, C, by the set, Y; extending partial homomorphisms in conjunction with determining lift from a homomorphism, h', to an
algebra, \( D \), or factor algebra, \( C/Y \); as well as some more trivial tools. Thus, in order to complete a computational model for rank approximation, the coding of some additional tools is necessary.

Finally, with the problem defined and the goals decided upon the next step is to discuss the choice of programming language. Upon inspection of the goals, the tools and structures needed for the type of system to be created, the logic programming language, Prolog, is the language of choice. The following section presents the reasons for this decision.

### 2.2 Logic Programming and Prolog

Logic programming is programming with the use of languages based on the foundations of predicate logic, as opposed to the more traditional, structured programming languages (e.g. C or Pascal), or object-oriented programming languages (e.g. C++ or Java). Prolog [12] is the most popular logic programming language. It is based on a subset of possible formulae, called Horn clauses, from first-order predicate calculus (FOPL). In this section of Chapter 2, the advantages of logic programming, specifically Prolog, as a programming language are discussed. The section is divided into six subsections, each representing a particular advantage of Prolog. Within each subsection the general advantage is given followed by an explanation of how this relates with the particular goal of the research.

One of the original motivations for doing this research is the idea that the language of universal algebra may be viewed in terms of a formal language. That is, in a general
sense, universal algebra can be seen to have a syntax (rules), semantics (meaning), and a lexicon (facts). For example, there is a rule of English language that says, “a statement is a sentence if it is comprised of a noun phrase followed by a verb phrase.” Similarly, it can be said that universal algebra has the rule, “an algebra, $\mathbf{B} = \langle B, f^B \rangle$, is said to be a subalgebra of an algebra, $\mathbf{M} = \langle M, f^M \rangle$, if $B$, the universe of $\mathbf{B}$, is a subset of $M$, and $f^B$ is closed under the operation of $\mathbf{M}$, $f^M$. Also, universal algebra has semantics, the algebras, $\mathbf{M}$ and $\mathbf{B}$, could be examples of various different types of algebras. In this case, it is known that $\mathbf{M}$ and $\mathbf{B}$ are mono-urnary algebras, based on the context of the thesis and the way the algebras are presented within the sentence. Furthermore, in English, a sentence is made up of words form a lexicon, where a lexicon represents the database of all known English words. With regards to the statement above, there may be a lexicon containing objects which are viable elements of an algebra and/or a lexicon of known algebras. Specific to the research of this thesis, universal algebra can be seen to have a lexicon of homomorphisms with known rank. Prolog is a logic based programming language that is used in some areas of Artificial Intelligence, in particular Natural Language Processing. Natural Language Processing is loosely defined as the study of natural human languages using computers. Taking in to consideration the comparison of universal algebra to formal languages mentioned, the use of Prolog for the purposes of this thesis seems a natural fit. With the idea that Prolog would be the right choice for this thesis, a deeper investigation into other advantages of its use follows.

Prolog allows for the manipulation of symbolic data as oppose to numeric data. This
is done through the use of data types such as atoms, predicates, lists and list structures coupled with unbound variables and unification. This use of symbolic processing in Prolog is less restrictive than numeric computation languages. Great freedom is allowed in the choice of operations that can be performed and the elements that can be used in these operations. What the symbolic processing capabilities of Prolog gives a programmer is qualitative representation of knowledge. Algebra is abstract; the elements and operations done in universal algebra reside on the symbolic level and thus are suited to symbolic representation. So, in relation to the problem analysis of the first section, Prolog is an advantageous programming language for its capability to represent the structures of the algebra and homomorphism symbolically.

Prolog programs are basically a database of rules and facts given within a specific domain. The idea is to query the program and based on these rules and facts, Prolog answers the query by possibly generating new knowledge in the domain. The whole process can be looked at as database manipulation on a relatively small scale. One of the advantages of this fact is that Prolog allows you to describe general ideas while at the same time define specific situations. Furthermore, this allows code to be compact and to the point, since rules are generally small and geared toward specific goals. Also, because Prolog rules tend to be small and variables are locally quantified, any changes that need to be made are local. As mentioned above, calculating rank is also based on a set of rules and facts. The rules being the steps involved in finding the rank of a homomorphism. The facts are a store (database) of homomorphisms with known rank. Initially, the database
contains only projections. Homomorphisms whose ranks are discovered are added to the
database.

One of the major program control constructs of Prolog is recursion. What this means
is that the predicates written in Prolog are normally defined recursively. With recursion
one can represent the usual constructs, \textit{for loops} and \textit{if-then-else statements}, as well as
make it easier traversing through such data types as the list. For example, in traversing
a list with recursion, the size of the list need not be known, which is not necessarily true
of the above constructs. Moreover, recursion in Prolog is more readable, it is easy to
follow where a predicate is going and what its goal is. Since recursion is a more concise
construct, it makes it easier to analyze for complexity. With Prolog the rank function
can be written recursively taking advantage of the rank definition’s built in recursion.

Another major drawing point of Prolog is its ability to move backwards, or backtrack,
through code. In other words, Prolog has the ability to search the entire proof tree for all
possible outcomes of a query by redoing earlier computations and trying other solutions.
Among other things, it allows the programmer to get more out of one predicate with
minimal extra work done. In calculating rank it is necessary to find all the subalgebras
of a given algebra at various points. With backtracking available, one need only write
code that will generate an arbitrary subalgebra. The backtracking feature makes possible
finding all the arbitrary subalgebras using this code.

One control construct used by Prolog, recursion, has already been mentioned. An-
other control construct in Prolog is the pattern-directed search. The control this exerts
comes in the form of the built in unification and depth-first search algorithm. By correctly ordering predicates one has control over where the search goes and where variable unification takes place. Furthermore, Prolog offers some procedural features such as `assert`, `fail`, and the cut, denoted by an exclamation mark, (!). The clause `assert` allows the programmer to add new predicates to the database during runtime. The `fail` causes the code to backtrack at any point in the code, and the cut controls the path of the search tree while backtracking. Control is a very important aspect when calculating rank, in a few different ways, which is why the flexibility of control offered by Prolog is attractive. First of all, when choosing homomorphisms for a set, $Y$, it is necessary to start with the homomorphisms of lowest rank and work up. Second, one needs to be able to add new homomorphisms to the database at runtime before trying the next homomorphism. The search tree for the rank function is deeply nested and can become rather large when the number of subalgebras grows quickly. There are two situations to control when travelling through the search tree of rank. One, if at any time something `fails` to be true about a subalgebra, the whole program should `fail`. Two, if a particular homomorphism extension, $h'$, or set of homomorphisms, $Y$, `fail`, then move on to the next one. For all these instances Prolog offers control devices to cope with them.

For all these reasons the best choice of programming language for this research is Prolog. Now, to introduce the code and explain the inner workings of its particulars.
2.3 The Computational Model

Section 4 of Chapter 2 describes the final incarnation of code that constitutes the computational model. The model approximates the rank of homomorphisms on algebras. Not all aspects of the model are presented here, for a complete look at the code refer to Appendix B. The first part of this section gives an introduction of the datastructures which represent algebras and homomorphisms. In the second part, a discussion of the database of homomorphisms with known rank is given. The final three parts of this section present the main body of code, rank/6. These three sections are divided into three major themes of rank/6, the use of recursion, the use of bagof/3 and setof/3 for all possible solutions, and the use of the cut for control of flow.

As discussed in the previous sections of this chapter, the basic building blocks of the rank definition are the algebra and the homomorphism. It was decided that the best way to represent these structures is symbolically. In Prolog this means the use of functors, lists or a combination of the two. For this model an algebra is represented by a functor with two arguments, called algebra/2. The first argument of algebra/2, which represents the universe of the algebra, is a list containing the elements of the universe, where each element is also given as a list. The second argument is a list of lists representing the unary functions of the algebra. Each sublist in the second argument represents one unary function on the algebra. A function list is made up of one or more instances of the functor, f/2, one for each element in the universe. The functor, f/2, has two arguments, both elements of the underlying universe of the algebra. The first argument is the element
being mapped and the second is the image of the first after applying the function. For example, algebras $M_1$ and $P_1$, given in Figure 2-1, are represented as follows: For $M_1$,

\[
\text{algebra}([[0], [a], [b], [c]], [[f([0], [0]), f([a], [0]), f([b], [0]), f([c], [a])]])
\]

and for $P_1$,

\[
\text{algebra}([[0], [a], [b]], [[f([0], [0]), f([a], [0]), f([b], [a])], [f([0], [a]), f([a], [b]), f([b], [b])]])
\]

To show the reasoning for representing elements of the algebra as lists, let us look at another example. Given algebra $M_3$ with universe $M_3 = \{0, a\}$, the algebra $(M_3)^2$ has universe $(M_3)^2 = \{(0, 0), (0, a), (a, 0), (a, a)\}$. Both algebras are shown in Figure 2-2, and the Prolog representation of $(M_3)^2$ in this model is

\[
\text{algebra}([[0, 0], [0, a], [a, 0], [a, a]],
\quad [[f([0, 0], [0, 0]), f([0, a], [0, 0]), f([a, 0], [0, 0]), f([a, a], [0, 0])]])
\]
For the computational model of this thesis, a homomorphism is represented by function \texttt{homomorphism/5}. Generally, the data structure for a homomorphism from an algebra, \texttt{AlgB}, to an algebra, \texttt{AlgM}, looks like

\begin{equation}
\text{homomorphism}(\text{Rank}, K, \text{AlgB}, \text{AlgM}, \text{Homo}),
\end{equation}

where \texttt{AlgB} is a subalgebra of algebra, \((\text{AlgM})^n\), for some finite, nonnegative \(n\), and arguments \texttt{Rank}, \(K\), and \texttt{Homo} are as follows. The variable, \texttt{Rank}, is the \(K\)th approximation of the rank of the homomorphism, \texttt{Homo}, where \(K\) represents a fixed finite integer. Variable \texttt{Homo} is a list containing two other lists. The first list in \texttt{HOMO} represents the domain of the homomorphism and the second is a list of two-argument functions, \texttt{h/2}, which represent the image, after applying the homomorphism, of each element in the domain. As an example, suppose we have a homomorphism from algebra, \((M_3)^2\), to algebra, \(M_3\), which maps every element in \((M_3)^2\) to the element, 0, in the universe, \(M_3\). Then, the variable
Homo would bind with the list,

$$[[0,0], [0,a], [a,0], [a,a]], [h([0,0],[0]), h([0,a],[0]), h([a,0],[0]), h([a,a],[0])].$$

The inclusion of the variable $K$ as an argument of `homomorphism/5` is necessary for the computational model. The $K^{th}$ approximation of the rank of a homomorphism is dependent on $K$, so the value of $K$, for any given homomorphism, is needed to make correct conclusions about reported ranks.

Recall, the approximation of the rank of an algebraic operation is based on the rank of other algebraic operations. Thus, it is necessary to create and maintain a database of homomorphisms with known rank. The database consists of assertions which are provided by headless predicates, each of the form

$$\text{homomorphism}(\text{Rank}, K, \text{AlgB}, \text{AlgM}, \text{Homo}).$$

Furthermore, the definition of rank, given in Chapter 1, states that $\text{rank}(h) \leq 0$ if and only if $h$ is a projection. This gives the initial set of homomorphisms for the database onto which more will be added. The database consists of two directories, `MUDatabase` and `BUDatabase`, the latter being the database of homomorphisms on bi-unary algebras, and the former the database of homomorphisms on mono-unary algebras. Each directory consists of multiple files containing the homomorphisms on particular algebras. The files are named for the algebra with the Prolog extension `pl`, since it is necessary to consult
the appropriate file. The predicate, make_projections/4, is used to create, for a fixed algebra, the initial database of projections, all of which have rank less than or equal to zero. The projections being used are those from all the subalgebras of finite powers of an algebra, up to a fixed size, to the algebra.

The code for make_projections/4 is as follows.

```prolog
% make_projections(+ALG_M,+I,+K,+FILE)
% Create a database of projections from subalgebras of finite powers of algebra, ALG_M, to ALG_M, for 1 <= I <= K. Store the database as a file with filename, FILE.

make_projections(algebra(M,FM),I,K,File) :-
    open(File,write,Stream),
    list_to_ord_set(M,OM),
    ord_functions(FM,OFM),
    for_each_i(algebra(OM,OFM),K,I,Stream),
    close(Stream).

% for_each_i(+ALG_M,+K,+I,+STREAM)
% For each integer, I, from 1 to some integer, K, construct the Ith power of algebra, ALG_M, and collect all the subalgebras of the resulting algebra.
for_each_i(Algebra_M,K,I,Stream) :-
    I=<K, !,
    make_M_to_the_n(Algebra_M,I,Algebra_Mi),
    bagof(Algebra_B,subalgebra(Algebra_B,Algebra_Mi),Subalgebras),
    for_each_sub(Subalgebras,Algebra_M,I,Stream),
    I1 is I+1,
    for_each_i(Algebra_M,K,I1,Stream).
for_each_i(Algebra_M,K,I,Stream).

% for_each_sub(+SUBS,+ALG_M,+I,+STREAM)
% Recursively traverse the list of algebras, SUBS.
for_each_sub([],Algebra_M,I,Stream).
```

for_each_sub([Alg_B|Algs_B], Algebra_M, I, Stream) :-
    for_each_j(Algebra_M, I, Alg_B, 1, Stream),
    for_each_sub(Algs_B, Algebra_M, I, Stream).

for_each_j(+ALG_M,+I,+ALG_MI,+J,+STREAM) :-
    for_each_j(+ALG_M,+I,+ALG_MI,+J,+STREAM),
    projections(Algebra_Mi,J,Homomorphism),
    write(Stream,homomorphism(0,'K',Algebra_Mi,Algebra_M,Homomorphism)),
    format(Stream,``w'',[.']),
    format(Stream,``n'',[1]),
    J1 is J+1,
    for_each_j(Algebra_M,I,Algebra_Mi,J1,Stream).

projections(algebra([],Functionlist),J,[[0],[0]]) .
projections(algebra([X !Xs],Functionlist),J,[[X|Rest1],[h(X,Y)|Rest]]) :-
    proj(J,X,Y),
    projections(algebra(Xs,Functionlist),J,[Rest1,Rest]) .

% for_each_j(+ALG_M,+I,+ALG_MI,+J,+STREAM)
% For each integer, J, from 1 to integer, I, construct the list
% of projections from an algebra, ALG_MI, to algebra, ALG_M,
% write them onto the stream, STREAM, as clauses of the
% predicate, homomorphism/5.

for_each_j(Algebra_M,I,Algebra_Mi,J,Stream) :-
    J=<I, !,
    projections(Algebra_Mi,J,Homomorphism),
    write(Stream,homomorphism(0,'K',Algebra_Mi,Algebra_M,Homomorphism)),
    format(Stream,``w'',[.']),
    format(Stream,``n'',[1]),
    format(Stream,``-n'',[1]),
    J1 is J+1,
    for_each_j(Algebra_M,I,Algebra_Mi,J1,Stream).

% projections(+ALG,+J,-PROJ)
% Argument, PROJ, binds with the homomorphism list created by
% taking the Jth projection of each element a in an algebra, ALG.

There are 4 arguments for this predicate, ALG_M, I, K, and FILE, all of which are bound
variables, meaning the 4 variables of this predicate are expected to be input by the user.
The variable, ALG_M, represents the underlying algebra. Variable FILE is the particular
file which the part of the database for ALG_M is stored in. For example, the database for
algebra M_1 of Figure 2-1, is stored in file `MUDatabase/algebraM1.pl'.
contains information about homomorphisms on subalgebras up to a finite power of an algebra. The variable, $K$, represents the finite power, which may be different from algebra to algebra. In the case of projections, the variable $K$ remains an unbound variable in the database, which binds to any value of $K$ used. Projections are of rank 0 no matter what value of $K$, by the definition of rank. Finally, the variable, $I$, is used as a counter up to $K$, which usually begins as 1. As an example of a database entry, consider again the algebra $M_1$ of Figure 2-1, and algebra $B_{11}$ a subalgebra of $M_1^2$. Let algebra $B_{11}$ be defined as follows,

$$algebra([[0,0],[a,b]], [[f([0,0],[0,0]),f([a,b],[0,0])]]),$$

where the first projection, $\pi_1$, from $B_{11}$ to $M_1$ is defined as

$$[[[0,0],[a,b]], [h([0,0],[0]),h([a,b],[a])]].$$

The entry in the database file 'MUDatabase/algebraM1.pl' looks like this:

$$\text{homomorphism}(1,K,\text{algebra}([0,0],[a,b]),\text{algebra}([[0,0],[a,b]],[[f([0,0],[0,0]),f([a,b],[0,0])]]),\text{algebra}([[0],[a],[b],[c]],[[f([0],[0]),f([a],[0]),f([b],[0]),f([c],[a])]]),)$$
So far, in this section, the basic datastructures used in the computational model and the code used for the creation of the database have been covered. Consider this the setup to the main programming that needs to be done. For the rest of the section a detailed look at the crux of the coding, predicate \texttt{rank/6}, is presented.

Now begins a comprehensive study of the main predicate, \texttt{rank/6}. It begins with an introduction to the predicate and its purpose in this thesis followed by a presentation of the variables in the arguments and the subroutines of \texttt{rank/6}. Following this is a look at three of the more interesting techniques used in the body of the predicate; these include the use of recursion, the built-in predicates \texttt{bagof/3} and \texttt{setof/3} for all solutions, and the use of ! and \texttt{fail} for control over the flow.

The predicate \texttt{rank/6} is used to approximate the rank of the algebraic operations of an algebra. The predicate \texttt{rank(Homo,Sn,K,N,File,Rank)} has six arguments, where \texttt{Homo}, \texttt{Sn}, \texttt{K}, \texttt{N}, and \texttt{File} are input and \texttt{Rank} is output. The argument \texttt{Homo} is itself a predicate representing the homomorphism in question, \texttt{homomorphism(Rank,K,AlgB,AlgM,H)}, which is a homomorphism, \texttt{H}, from the algebra \texttt{AlgB} to the algebra \texttt{AlgM}. Argument \texttt{Sn} is a positive integer that represents the power of the algebra \texttt{AlgM} for which \texttt{AlgB} is a subalgebra. The variable \texttt{N} is also a predetermined integer value which is an upper bound. The variable \texttt{K} is a finite, nonnegative integer representing the value of \texttt{K} in the \texttt{Kth} approximation of rank definition. Variable \texttt{File} is the file in which the database of alge-
braic operations on algebra $\text{AlgM}$ with known rank are stored. Finally, when a value for rank is approximated it binds with the variable $\text{Rank}$ and is returned as output. When a homomorphism, $H$, of this model is of rank $\text{Rank}$, it represents the $K$th approximate rank, which satisfies $\text{rank}(H) \leq \text{Rank}$.

A skeletal look at how the model approximates the rank of an algebraic operation, using the Prolog notation of variables, follows. Given the arguments

$$\text{homomorphism}(\text{Rank}, K, \text{AlgB}, \text{AlgM}, H),$$

$\text{Sn}$, $\text{N}$, and $\text{File}$, discussed above, the rank of homomorphism, $H$, is approximated through the following steps. For each nonnegative integer up to a fixed, finite $K$, embed an algebra, $\text{AlgB}$, into an algebra called $\text{AlgBprime}$, by repetition of $K$ coordinates for all possible repetitions. For each algebra, $\text{AlgBprime}$, a subalgebra of algebra, $\text{AlgM}$, to the power of $\text{Sn}+K$, where homomorphism, $Hprime$, is the extension of $H$, generate all algebras, $\text{AlgD}$, such that $\text{AlgBprime}$ is a subalgebra of $\text{AlgD}$, which is a subalgebra of $\text{AlgM}$ to the power of $\text{Sn}+K$. For each algebra, $\text{AlgD}$, where $Hprime$ lifts to $\text{AlgD}$, collect all possible sets, $Y$, of algebraic operations on $\text{AlgM}$ with known rank, such that the size of each $Y$ is less than or equal to $N$. Furthermore, for all $\text{AlgD}$, generate all algebras, $\text{AlgC}$, such that $\text{AlgBprime}$ is a subalgebra of $\text{AlgC}$ a subalgebra of $\text{AlgD}$. For each algebra, $\text{AlgC}$, find a set, $Y$, such that $Y$ separates $\text{AlgBprime}$ and $Y$ lifts to the factor algebra of $\text{AlgC}$ with respect to $Y$, called $\text{AlgCmodY}$. The rank of homomorphism, $H$, is $\text{Rank}$, which is equal to $\text{MaxRank}+1$, where $\text{MaxRank}$ is the rank of the homomorphism,
g, with the highest rank in the set of homomorphisms, Y. Beyond \texttt{rank/6}, the body of code is nested seven predicates deep. Those predicates are \texttt{rank/9}, \texttt{for._each_k/9}, \texttt{for._each_part/9}, \texttt{for._each_D/8}, \texttt{for._each_C/7}, \texttt{there._exists_Y/7}, and \texttt{for._each_g/3}.

The rank is passed throughout the code as an accumulator pair of \texttt{Rank} and \texttt{MaxRank}, where \texttt{MaxRank} changes at the base case in \texttt{for._each_g/3}, if the rank of any g is bigger than the current \texttt{MaxRank}. Although we do not go into great detail for each predicate, we look at the uses of recursion, \texttt{bagof/3}, and the cut.

\texttt{Rank} is defined recursively on the rank of simpler homomorphisms. Recursion is used in three ways throughout the body of \texttt{rank/6}. Implicitly, as in the mathematical definition of rank, the rank of a homomorphism, H, is based on the ranks of the homomorphisms, g, in the set, Y, where \texttt{rank(H) \leq MaxRank+1}, such that the rank of each g in Y is less than or equal to \texttt{MaxRank}. Explicitly, in predicate \texttt{for._each_k/9}, recursion is used as a for loop for each I such that 0 is less than or equal to I less than or equal to K. Furthermore, recursion is used in predicates, \texttt{for._each_part/9}, \texttt{for._each_D/8}, \texttt{for._each_C/7}, \texttt{there._exists_Y/7}, and \texttt{for._each_g/3} as a means of list traversal, with base case empty list for each.

For the predicates \texttt{for._each_D/8} and \texttt{for._each_C/7} it is necessary to generate a list of all the appropriate subalgebras, \texttt{AlgD} and \texttt{AlgC}, respectively. Mathematically, we are moving from quantification over algebras to quantification over subalgebras. Logically, we are moving from FOPL to higher-order predicate logic. To do this, the built-in predicate \texttt{bagof/3} and the predicate \texttt{subalgebra/3} specially written for this thesis are
used. Predicate `subalgebra(A,B,C)` takes algebras A and C, where A is a subalgebra of C, as input, and finds an instance of an algebra which is a subalgebra of C and has A as a subalgebra, and binds it to output variable, B. The built-in predicate, `bagof/3`, is of the form

```
bagof(Template,Goal,List),
```

and is read as the list, List, of all instances of Template satisfied by Goal. As an example, consider the case of generating all algebras, AlgD, such that AlgBprime is a subalgebra of AlgD, which is a subalgebra of AlgM to the power of Sn+K. In order to do this, the predicate `for_each_part/9`, which is written

```
% for_each_part(+PARTS,+HOMO,+ALG_B,+ALG_M,+ALG_MnK,+K,+N,+MAXR,-R)
% If all partitions in the list argument, PARTS, are exhausted,
% then argument, R, binds with MAXR, the maximum rank of
% homomorphism, HOMO. Otherwise, recursively traverse list, PARTS,
% to approximate the rank of homomorphism, HOMO.
```

```
for_each_part([Part|Parts],Homo,Alg_B,Alg_M,Alg_MnK,K,N,MR,R) :-
    embed_B(Alg_B,Part,algebra(Bprm,FBprm)),
    list_to_ord_set(Bprm,OBprm),
    ord_functions(FBprm,OFBprm),
    is_algebra(algebra(OBprm,OFBprm)),
    subalgebra(algebra(OBprm,OFBprm),Alg_MnK),
    make_hprime(Homo,algebra(OBprm,OFBprm),Hprm),
    is_homomorphism(homomorphism(R,K,algebra(OBprm,OFBprm),Alg_M,Hprm),
    bagof(Alg_D,subalgebra(algebra(OBprm,OFBprm),Alg_D,Alg_MnK),SubAlgs_D),
    !, for_each_D(SubAlgs_D,Hprm,Alg_M,algebra(OBprm,OFBprm),K,N,MR,TR),
```

calls `bagof/3` to generate the list, SubAlgs_D, of all the subalgebras between an algebra, `algebra(OBprm,OFBprm)`, and algebra, `Alg_MnK`, by repeated calls to `subalgebra/3`. 

36
For predicate \texttt{there\_exists\_Y/7}, a list of all possible sets, \( Y \), of homomorphisms, \( g \), from algebra, \texttt{AlgD}, to algebra, \texttt{AlgM}, is needed. To accomplish this goal there is a call to predicate, \texttt{get\_Ys/5}, in the body of \texttt{for\_each\_D/8}. In \texttt{get\_Ys/5}, shown below, there are two calls to \texttt{setof/3}. The built-in predicate \texttt{setof/3} consists of a call to \texttt{bagof/3} followed by a call to the built-in predicate, \texttt{sort/2}. Predicate, \texttt{sort/2}, sorts the list, \texttt{List}, from \texttt{bagof/3}, and removes duplicate entries from the list, resulting in an ordered set.

\[
\% \text{get\_Ys(+ALG\_M,+ALG\_MN,+K,+N,-LISTOFYS)}
\% \text{When using get\_Ys/5, it is assumed that the database of algebraic}
\% \text{operations of an algebra, ALG\_M, with known rank are asserted as}
\% \text{facts in the system. All instances of homomorphism/5 with}
\% \text{arguments K, ALG\_M, and ALG\_MN are collected in an ordered set.}
\% \text{The order of this set is based on the rank of the homomorphisms.}
\% \text{Reverse the order of this set then collect all instances of}
\% \text{sublists of the set in an ordered set. Each sublist is ordered}
\% \text{from maximum rank to minimum. Finally, remove all the sublists}
\% \text{that are strictly larger than N.}
\]

\[
\text{get\_Ys(M,Mn,K,N,Ys) :-}
\text{setof(homomorphism(R,K,Mn,M,H),homomorphism(R,K,Mn,M,H),Gs)},
\text{reverse(Gs,RGs)},
\text{setof(G,sublist(G,RGs),Zs)},
\text{sizeof\_g\_lessthan\_N(Zs,N,Ys)).}
\]

Here, the first call to \texttt{setof/3} collects all the homomorphisms from algebra \( Mn \) to algebra \( M \), for some \( K \), into a list, \( Gs \), ordered by rank. The call to \texttt{reverse/2} reverses the order of the list so that homomorphisms with maximum rank come first (i.e. decreasing order on rank). The built-in predicate \texttt{sublist/2} maintains the order previously imposed on the original list, thus the second call to \texttt{setof/3} generates all the lists, \( G \), where \( G \) is a sublist of \( Gs \), and stores them as the ordered list of lists \( Zs \). The order of \( Zs \) is increasing, based
on the ranks of the first homomorphism in each sublist contained in Zs. Therefore, all the sublists containing homomorphisms with a maximum of rank zero come first, followed by those with a maximum of rank one, and so on. Why do it like this? If there is a Y containing projections only, that separates \textit{AlgBprime} and lifts to \textit{AlgCmodY}, then it does not matter if a Y with a rank one homomorphism exists that does the same thing. Within each loop, one wants the minimum rank possible.

Why use \texttt{bagof/3} and \texttt{setof/3} instead using the backtracking feature explicitly? Backtracking is tough to control in the cases mentioned above. There are cases within rank where we want a \texttt{failure} in a sub-routine to result in a \texttt{failure} in the main routine (discussed in the next section). It is not so easy to differentiate between these \texttt{failures} when they occur on the same level of programming. By using the predicates mentioned, we make use of backtracking explicitly by collecting all the solutions in a list and then use recursion to test each of the solutions.

Next, we discuss the use of the cut, \texttt{!/0}, \texttt{fail/0}, and \texttt{assert/1}, for control over the flow of the code. An important aspect of any computer programming is control over the flow of the code. In particular, this means being able to prune fruitless branches from the search tree. This point becomes very apparent when coding the problem of approximating rank. Even for relatively small algebras, calculating the rank of their algebraic operations creates a large search tree in a short time. The reason for the rapid growth of the search tree is the potentially large number of subalgebras generated. Recall, in the body of the predicate, \texttt{rank/6}, the generation of all subalgebras, \texttt{AlgD}, between
some algebra AlgBprime and algebra AlgMSnplusK is a necessity. Then, for each of those subalgebras, Prolog generates all the subalgebras, AlgC, between algebra AlgBprime and algebra AlgD. For this reason it is necessary to incorporate the built-in predicates, !/0 and fail/0, to trim the search tree.

There are two types of cut used in Prolog, as discussed by Richard O'Keefe in his text, *The Craft of Prolog* [16]. The two cuts in question are called red cuts and grue cuts. In simple terms, red cuts cause Prolog not to consider later clauses of a goal, while grue cuts keep Prolog from backtracking through the search tree unnecessarily, looking for other solutions. The predicate fail/0, when used, causes Prolog to fail and initiate backtracking to find alternative solutions. Both types of cut and the fail/0 are used in the body of the code of rank/6, in order to control flow. For clarity, different placement points for the different cuts are used in the code of this research. Red cuts are placed at the end of a line, and grue cuts at the beginning of a line. A presentation of three examples from the code, one of the red cut, one of the grue cut, and one of the fail in conjunction with a cut, is made next.

Recall that the quantification in the definition of rank consists in part of the statement that for all subalgebras, D, of algebra M^{n+k} and for all subalgebras, C, of D, where h' lifts to D, there exists a set, Y, of homomorphisms from D to M such that |Y| ≤ N, h' lifts to C/Y, and rank(g|_C) < α, for all g ∈ Y. In the code, this statement is represented by the three predicates, for_each_D/8, for_each_C/7, and there_exists_Y/7, each controlling the search tree in a different way. The first predicate, for_each_D/8, represents the line,
"...for all subalgebras, $D$, of algebra, $M^{n+k}$, and for all subalgebras, $C$, of $D$, where $h'$ lifts to $D$, ...", and is presented here.

\%
for_each_D(+ALGS_D,+HPRIME,+ALG_M,+ALG_BPRIME,+K,+N,+MAXRANK,-RANK)
% If all the subalgebras in the list, ALGS_D, are successfully
% exhausted, RANK binds with argument, MAXRANK. Otherwise,
% recursively traverse the list, ALGS_D, looking for only those
% algebras in the list which the homomorphism, HPRIME, lift to.
% Finally, if HPRIME does not lift to an algebra in the list,
% ALGS_D, move on to the next algebra.

for_each_D([algebra(D,FD)|Alg_Ds],Hprm,Alg_M,Alg_Bprm,K,N,MR,R) :-
    list_to_ord_set(D,OD),
    ord_functions(FD,OFD),
    lifts(Hprm,algebra(OD,OFD),Alg_M),
    get_Ys(Alg_M,algebra(OD,OFD),K,N,Ys),
    bagof(Alg_C,subalgebra(Alg_Bprm,Alg_C,algebra(OD,OFD)),SubAlgs_C),
    !, for_each_C(SubAlgs_C,Hprm,Alg_M,Alg_Bprm,Ys,MR,TR),
    for_each_D(Alg_Ds,Hprm,Alg_M,Alg_Bprm,K,N,TR,R).
for_each_D([Alg_D|Alg_Ds],Hprm,Alg_M,Alg_Bprm,K,N,MR,R) :-
    for_each_D(Alg_Ds,Hprm,Alg_M,Alg_Bprm,K,N,MR,R).

The cut appears at the beginning of the sixth line, in the tail of the second clause. In this
case, if the call to predicate, lifts(Hprm,algebra(OD,OFD),Alg_M), fails, then Prolog
proceeds to the third clause of for_each_D/8 and moves on to the next algebra, Alg_D,
in the list. However, if lifts(Hprm,algebra(OD,OFD),Alg_M) succeeds, followed by a
fail in predicate, for_each_C/7, then for_each_D/8 will fail also. That is, Prolog will
not search any further algebras, Alg_D, in the list, as well as not move back to previously
searched algebras, by pruning all future solutions and proofs of the query.

The predicate for_each_C/7 consists of two clauses which traverse the list of subal-
gebras, $C$, passing each to there_exists_Y/7 for testing. In the case of for_each_C/7,
if there_exists_Y/7 fails on any given C, then no further solutions to the query exist.

Here one does not want Prolog to backtrack to a previous subalgebra, C, and try again.

The placement of a cut, in the second clause, in front of the call to there_exists_Y/7, achieves this goal by pruning the search tree of any other proofs.

% for_each_C(+ALGS_C,+HPRIME,+ALG_M,+ALG_BPRIME,+YS,+MAXRANK,-RANK)
% If all subalgebras in the list, ALGS_C, are successfully
% exhausted, RANK binds with the maximum rank, MAXRANK. Otherwise,
% recursively traverse the list, ALGS_C.

for_each_C([],Hprm,Alg_M,Alg_Bprm,Ys,R,R).
for_each_C([algebra(C,FC)|Alg_Cs],Hprm,Alg_M,Alg_Bprm,Ys,MR,R) :-
    list_to_ord_set(C,OC),
    ord_functions(FC,OFCC),
    !, there.exists_Y(Ys,algebra(OC,OFCC),Alg_M,Alg_Bprm,Hprm,MR,TR),
    for_each_C(Alg_Cs,Hprm,Alg_M,Alg_Bprm,Ys,TR,R).

For the final predicate, there_exists_Y/7, one wants to test each set, Y, until Prolog succeeds or the sets Y are exhausted. Once Prolog finds a Y that succeeds the search may continue on to the next C. If all the choices for Y fail, then the entire search must fail. To accomplish this, a cut is placed before a call to fail/0, in the first clause. This clause is the base case for this predicate and it indicates that Prolog has reached the end of the list without finding a solution to this branch of the search tree. So, Prolog is forced to fail in order to back out of the query.

% there_exists_Y(+YS,+ALG_C,+ALG_M,+ALG_BPRIME,+HPRIME,+RANK,-NEWRANK)
% Recursively traverse the list argument, YS, of sets of
% homomorphisms with known rank, searching for a set which
% satisfies all the predicates in the tail of the second clause.
% If a set does not satisfy all the predicates, move on to the
% next set using the third clause. If no such set exists, cause
% a fail of the whole predicate by the first clause.
there_exists_Y([Y|Ys],Alg_C,Alg_M,Alg_Bprm,Hprm,R,NewR) :-
  separates(Y,Alg_Bprm,Fact_Alg),
  factor_algebra(Alg_C,Y,Alg_Bprm,Fact_Alg,algebra(CmodY,FCmodY)),
  list_to_ord_set(CmodY,OCmodY),
  ord_functions(FCmodY,OFmodY),
  is_algebra(algebra(OCmodY,OFmodY)),
  lifts(Hprm,algebra(OCmodY,OFmodY),Alg_M),
  for_each_g(Y,R,NewR).
there_exists_Y([Y|Ys],Alg_C,Alg_M,Alg_Bprm,Hprm,R,NewR) :-

This concludes our look at the main predicate of the research, rank/6. To view rank/6
in its entirety consult Appendix B. The appendix also contains the rest of the code written
to facilitate the approximation of rank. These include; make_M_to_the_n/3, for generating
power algebras; subalgebra/3, subalgebra/2, and gen_subalgebra/3, for testing
and creating subalgebras; homomorphism/4, for testing and creating homomorphisms,
as well as extending partial homomorphisms; partition_k/3, for use in generating all
the embeddings of an algebra; embed_B/3, for embedding by repetition of coordinates;
lifts/3, to test for lifting of a homomorphism; separates/3, to test for separation of an
algebra; factor_algebra/5, for generating a factor algebra; make_projections/4, de-
scribed above; and a handful of tools used specifically by the predicates above or shared
by a few. The last thing needed to be done in the presentation of the computational
model is a complexity analysis of the program.
2.4 Complexity Analysis

To analyze the code written for the approximation of rank we use a method called Big-Oh analysis. The definition,

\[ T(n) = O(f(n)) \] if there are constants \( c \) and \( n_0 \) such that \( T(n) \leq cf(n) \) when \( n \geq n_0 \),

is given by Mark Allen Weiss in his text, *Data Structures and Algorithm Analysis in C++* [21]. Complexity analysis is done on time and space related issues. In this thesis it is done on the running times of the code. The reason for this choice is discussed further on in this section but it is enough to mention here that time and space for the research are closely related due to the lists of subalgebras generated while calculating rank. Furthermore, space complexity is always bounded by time complexity. Some useful rules of Big-Oh analysis that are used in the analysis of the research are as follows. To calculate the running time of a predicate that loops, multiply the number of iterations of the loop by the running time of the statements in the tail of the predicate. The running time of nested predicates is the running time of the innermost predicate multiplied by the number of iterations of each outside predicate. The running time of consecutive statements in the tail of a predicate is equal to the maximum running time of each of the statements. The runtime of a predicate with multiple clauses is the runtime of the test of the clause, times the maximum running time of the statements inside the clauses. Because of the lack of a priori knowledge of the number of subalgebras of any given algebra, the analysis in
This study is of the worst-case scenarios. The number of subalgebras of a given algebra is bounded by the number of subsets of the underlying set of the algebra. Thus, for the purposes of this research the worst-case scenario is when every subset of an algebra represents a subalgebra.

This section presents the variables used in the complexity analysis of the code as well as a table consisting of each predicate and its Big-Oh analysis. Following this is an explanation of the results of the analysis done on predicates \texttt{make_projections/4} and \texttt{rank/6}. This section concludes with a discussion of the use of the worst-case scenario and the time versus space issues.

The variables used in the complexity analysis are based on the input variables of the predicate, \texttt{rank/6}. The approximation of the rank of an algebraic operation, \( h \), on an algebra \( M \), assumes the input of the following; a positive integer, \( n \); a subalgebra, \( B \), of algebra, \( M^n \); a nonnegative integer, \( k \); and a positive integer, \( N \). Let \( |M| = m \), the number of elements in the algebra, \( M \), then, it follows that the size of \( M^n \) is \( m^n \) and the size of \( B \) is also \( m^n \), in the worst-case. Furthermore, let \( l \) represent the number of functions in the function list of the algebra, \( M \). Table 2.1 contains the names of various predicates and their Big-Oh analysis.

The database of projections, which have rank 0, is created using the predicate, \texttt{make_projections/4}. This predicate generates the database, given an algebra, \( M \), of all the projections from \( M^I \) to \( M \) for \( I \leq K \). Let \( |M| = m \) and let \( K = n + k \), where \( n \) is a positive integer and \( k \) is a nonnegative integer. To determine the com-
Complexity Analysis

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Complexity</th>
<th>Complexity</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>embed/3</td>
<td>$O(n^2 \cdot k \cdot m^n)$</td>
<td>$O(N \cdot m^{2(n+k)})$</td>
<td>$O(l \cdot m^{3(n+k)})$</td>
</tr>
<tr>
<td>factor_algebra/5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gen_subalgebra/3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_Ys/4</td>
<td>$O\left(2^{m^2+1}\right)$</td>
<td>$O\left(l \cdot m^{4(n+k+1)}\right)$</td>
<td>$O\left(l \cdot m^{4(n+k+1)}\right)$</td>
</tr>
<tr>
<td>homomorphism/4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lifts/3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>make_hprime/4</td>
<td>$O(m^n)$</td>
<td>$O\left((n+k)^3 \cdot m^{n+k} \cdot 2^{m^{n+k}}\right)$</td>
<td>$O(l)$</td>
</tr>
<tr>
<td>make_projections/4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ord_functions/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>partition_k/3</td>
<td>$O\left(\frac{(k+n-1)!}{(n-1)! \cdot (k-1)!}\right)$</td>
<td>$O\left(N \cdot m^{2(n+k)}\right)$</td>
<td>$O\left(l \cdot m^{2(n+k)}\right)$</td>
</tr>
<tr>
<td>separates/3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subalgebra/2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Big-Oh analysis on selected predicates.

Complexity of `make_projections/4`, note that it consists of five nested loops, `for_each_i/4`, `for_each_sub/4`, `for_each_j/5`, `projections/3`, and `proj/3`, listed from outer most to inner most. Therefore, the running time of `make_projections/4`, is at most the running time of `proj/3` times the number of iterations of each of the outside predicates, `for_each_i/4`, `for_each_sub/4`, `for_each_j/5`, and `projections/3`. The function of `proj(J,Element,P)` is to find the Jth projection, P, where J is between 0 and n + k, of `Element`, an element of $M^{n+k}$. Referring to the code for `proj/3` below, the Jth projection of `Element` is found by traversing the list, `Element`, to the Jth position and binding the element there with variable, P.

```%
% proj(+N,+ELEMENT,-P)
% The argument, ELEMENT, is instantiated with an element list. If
% the integer, N, is strictly larger than the size of the list,
% 45
```
% ELEMENT, then proj/3 fails. Otherwise, argument, P, binds with
% the Nth projection of an element, ELEMENT.

proj(N, [], P) :- !, fail.
proj(1, [X|Rest], [X]).
proj(N, [X|Rest], P) :-
   N1 is N-1,
   proj(N1, Rest, P).

The cost of doing this is the cost of traversing the entire list, which is of size, $n+k$, in the
worst-case. Working our way from inside out, the number of iterations of projections/3
is equal to the number of elements in the algebra $M^i$ where $i$ is greater than or equal
to zero and less than or equal to $n+k$. Thus, at the worst-case, $|M^i| = m^{n+k}$, making
the number of iterations also $m^{n+k}$. For both, for_each_i/4 and for_each_j/5, the
number of iterations is at most $n+k$. The predicate for_each_sub/4 traverses a list of
all subalgebras of algebra, $M^i$, where $i$ is at most $n+k$. There are at most $2^{m^{n+k}} - 1$
possible subalgebras of $M^i$, therefore as many iterations of for_each_sub/4. The empty
set does not make a subalgebra in our view, thus we subtract 1, the case that none of
the elements are in the subalgebra. Therefore, make_projections/4 has a running time
of at most

$$(n+k) \cdot m^{n+k} \cdot (n+k)^2 \cdot (2^{m^{n+k}} - 1),$$

or

$$T(n, k, m) = O((n+k) \cdot m^{n+k} \cdot (n+k)^2 \cdot (2^{m^{n+k}}))$$

$$= O((n+k)^3 \cdot m^{n+k} \cdot (2^{m^{n+k}})).$$

46
The analysis of the predicate rank/6 is similar to that of make_projections/4 in that rank/6 consists of nested predicates also, with the inner most predicate for_each_g/3.

Thus, the running time of rank/6 is at most the running time of for_each_g/3 times the number of iterations of each of the predicates, there_exists_y/7, for_each_C/7, for_each_D/8, for_each_part/9, and for_each_k/8. The cost of running for_each_g/3 is equal to one times the cost of traversing a given list of homomorphisms, Y. The size of Y is at most \( N \), a predetermined integer value. So, the running time of for_each_g/3 is at most \( N \). The number of iterations of there_exists_y/7 is at most the cost of traversing a list of all the possible sublists of homomorphisms from an algebra, \( M^{n+k} \) to algebra, \( M \). The number of possible homomorphisms from \( M^{n+k} \) to \( M \), is \( m^{mn+k} \), since there are \( m \) choices for each of the \( mn+k \) elements of \( M^{n+k} \). Again, this is a worst-case scenario since it is not always true that all these possibilities make a homomorphism. To get the number of sublists of this list of all the homomorphisms, note that either each homomorphism is in a sublist or it is not, thus there are two choices for each. Assuming, in the worst-case, that \( N \) is greater than or equal to \( m^{mn+k} \) then the number of sublists is at most \( 2^m^{mn+k} \). Both the predicates, for_each_C/7 and for_each_D/8, traverse a list of subalgebras with an equal maximum size. The maximum number of subalgebras in the list is equal to the maximum number of subalgebras of an algebra, \( M^{n+k} \). The algebra \( M^{n+k} \) contains \( mn+k \) elements, each of which has two possibilities, either it is in any given subalgebra or it is not. Thus the maximum number of subalgebras is \( 2^{mn+k} - 1 \).

The number of iterations of the predicate for_each_part/9 is equal to the number of
ways of partitioning an integer $k$ into a list of size $n$, which equals

\[
\binom{k + n - 1}{n - 1}.
\]

Finally, the predicate, `for_each_k/8`, is really just a for loop from 0 to some $k$, where $k$ is a nonnegative integer. Thus `for_each_k/8` consists of $k + 1$ (because we start at zero) iterations. Therefore, the running time of `rank/6` is equal to

\[
T(n, k, m, N) = O\left( N \cdot 2^{m^{n+k}} \cdot (2^{m^{n+k}} - 1) \cdot (2^{m^{n+k}} - 1) \cdot \binom{k + n - 1}{n - 1} \cdot (k + 1) \right)
\]

\[
= O\left( N \cdot k \cdot \binom{k + n - 1}{n - 1} \cdot (2^{m^{n+k}})^2 \cdot 2^{m^{n+k}} \right)
\]

in the worst-case scenario.

It is worth taking the time right now to investigate the use of the worst-case scenario throughout the complexity analysis for this research. In some instances the number of subalgebras is grossly over estimated. It is assumed that it is the case that every subset of an algebra represents the underlying universe of a subalgebra of the algebra. In practice, the number of subsets that are actually subalgebras may vary greatly from algebra to algebra. In fact, in some cases there will be only one subalgebra. Let us look at two examples which show both ends of the range. Let $M_3$ and $M_{12}$ be the three-element, mono-uniary algebras given in Figure 2-3.
The algebra, $M_9$, has seven subalgebras, that is, one for every subset of the set $\{0, a, b\}$ minus the empty set, whereas, algebra, $M_{12}$, has only one subalgebra, which is the algebra itself. Furthermore, in analyzing $\text{rank}/6$, we assume the list of all possible sublists of homomorphisms from an algebra, $M^{n+k}$, to algebra, $M$, is of size, $2^{mn^{n+k}}$. This number also represents a possibly huge over estimation. As a matter of fact, there are three assumptions made in arriving at this figure which take the worst-case into account. First, that all possible mappings from one algebra to another are actually homomorphisms. Second, that all those homomorphisms appear in the database with a value for their rank. The third assumption is that the boundary, $N$, is larger than the number of homomorphisms. That is, that we do indeed want all the sublists, not a bounded number of them. So our complexity analysis overestimates our actual running times in most cases, but their is no alternative here without prior knowledge of the number of subalgebras of an algebra or the number of homomorphisms.

The space costs throughout the code is bounded by the expected time costs. That is, the time needed to consider all $C \leq D$ is proportional to the number of $C$ times the number of $D$, whereas the space required is proportional to the number of $D$ plus the
The largest number of C. The time complexity is strongly dependent on the time it takes to generate and test all the possible subalgebras for each run of the code. Similarly, the space complexity is based on the amount of memory that is spent on storing all the possible subalgebras during a given run. Thus, it is acceptable to restrict the analysis to one case.

2.5 Summary

To summarize, the presentation of the computational model consists of four distinct steps. The first breaks the problem down into an experimental design, in which the obstacles of programming particular problems are defined. The second step justifies the choice of programming language and discusses the languages merits with respect to the problem. The third step discusses the implementation of the code; that is, the creation of the computational model. Finally, the fourth step analyzes the complexity of the code.
Chapter 3

Results and Discussion

3.0 Introduction

Chapter 3 presents an analysis of the results obtained through running the computational model introduced in Chapter 2. This includes a study of the runtimes of the predicate, \texttt{make\_projections/4}, for a few mono-unary and bi-unary algebras. The results and runtimes of \texttt{rank/6}, are also discussed. The running of \texttt{rank/6} gives two types of results for analysis, timing and approximations of the rank. This thesis looks at mono-unary and bi-unary algebras specifically, with the mono-unary algebras labelled as \( M_i \), and the bi-unary algebras labelled \( P_j \), where \( i \) and \( j \) are finite positive integers greater or equal to one. The results are presented in a series of tables, accompanied by figures which give a representation of the relevant algebras.
3.1 make\_projections/4

Recall from Chapter 2, the worst-case time complexity of the four argument predicate, \texttt{make\_projections/4}, is $T(m, k) = O\left(k^3 \cdot m^k \cdot \left(2^{m^k}\right)\right)$, where $m$ is the size of a given algebra, say $M$, and $k$ is the power of an algebra, say $B$, where $B \leq M^n$. That is, $T(m, k) = O\left(f(m, k)\right)$, where $f(m, k) = k^3 \cdot m^k \cdot \left(2^{m^k}\right)$. This section looks at the actual times of running \texttt{make\_projections/4} on some mono-ary and bi-ary algebras of size up to and including $m = 4$. The times are presented in tables followed by a discussion of the findings of the author. The times in the tables represent a cumulative figure, that is, the runtime of \texttt{make\_projections/4} for some particular $k$, is the runtimes for each $i$, where $i = 1, 2, \ldots, k$. Times presented in the tables are based on an average over one hundred runs except where marked by an asterisk, *, which are taken from the time of one run. The times represent CPU time and are given in milliseconds (ms) as reported by the SICStus Prolog interpreter.

Figure 3-1 contains a pictorial representation of the one-element algebras, $M_2$ and $P_2$, where $M_2$ is mono-ary and $P_2$ bi-ary. As far as algebras are concerned, these two are rather uninteresting on their own. For the sake of looking at the complexity analysis though, with these two algebras the worst-case is the only case. That is, there exists one subalgebra for each power algebra, which is the worst-case scenario. Given that $m = 1$ for these algebras, the complexity is simplified to

$$T(m, k) = T(1, k) = O\left(k^3 \cdot 1^k \cdot \left(2^{1^k}\right)\right) = O\left(k^3\right).$$

52
So, for \( m = 1 \), \( f (m, k) = f (1, k) = k^3 \).

Table 3.1 contains the observed running time of the predicate, `make_projections/4`, on the two one-element algebras, for various values of \( k \). Notice that the CPU time for running the predicate on algebra, \( P_2 \), is slightly larger than that of the times of running on algebra, \( M_2 \). The only difference between these two algebras is that \( P_2 \) has one more function than \( M_2 \), but the analysis suggests that the running time of the predicate is independent of the number of functions in the algebra. Also presented is a column containing the CPU times for \( P_2 \) divided by the times for \( M_2 \), to compare the growth rate of the two. From these numbers, \( P_2 \) is approximately 1.35 times slower than \( M_2 \), for \( k = 10, 25, 50, 75, 100, 200 \), which would suggest there is a constant difference between the growth rates. This may be explained by the original analysis work. Recall, in doing the complexity analysis, constants are ignored, the important part is the comparative growth rates.

One way to test the validity of Big-Oh analysis is to divide the actual running time of a predicate by the estimated running time. For example, given that a predicate’s running time, \( T (n) \), has been analyzed as being \( O (f (n)) \), for some function, \( f (n) \), divide \( T (n) \) by \( f (n) \) for a range of \( n \). If the computed values converge to a positive constant, then \( f (n) \) is a good estimate for the running time. Also, if the values converge to zero, then
Table 3.1: Running times and complexity of make_projections/4 on algebras $M_2$ and $P_2$. Size $m = 1$. Only 1 subalgebra for each $k$.

$f(n)$ is an over-estimate for the running time, but still an acceptable one. Where as, if the values diverge, the estimate for the running time, $f(n)$, is an under-estimate and should be re-evaluated.

In the case of algebras, $M_2$ and $P_2$, the generic running time of the predicate, make_projections/4, is estimated to be $O(k^3)$. The values derived from dividing the actual CPU times of running the predicate, for various $k$, by $k^3$ are recorded in the last two columns of Table 3.1. The calculated values are converging towards zero at a slow
rate, which suggests the use of $k^2$ is a good although slight over-estimate. Furthermore, recall that the case of $m = 1$ represents one of the examples where all possibilities are explored, i.e. all the sublists exist as subalgebras, which suggests that the estimate for the running time, $O\left(k^3 \cdot m^k \cdot (2^{m^k})\right)$, of predicate, `make_projections/4`, is an acceptable estimate.

The diagrams contained in the figures, Figure 3-2, Figure 3-3, and Figure 3-4, represent all the 2-element mono-unary and bi-unary algebras, up to isomorphism. The estimate of the running time of `make_projections/4`, on these algebras is

$$T(m, k) = T(2, k) = O(f(2, k)) = O\left(k^3 \cdot 2^k \cdot (2^{2^k})\right)$$

for finite nonnegative integer, $k$. For each of these algebras, the time (in milliseconds), the number of subalgebras (SA), and the times divided by $f(2, k)$, for each $k$, are recorded in Table 3.2 and Table 3.3. It can be seen in both tables the runtimes grow quickly as the value of $k$ goes from 1 to 4. From the function, $f(2, k)$, it is expected that the runtimes are strongly dependent on $k$. The times grow so quickly because of the $2^{2^k} - 1$ sublists which must be tested for each $k$. Within each $k$, it is shown that the runtimes are strongly dependent on the actual number of these sublists which are subuniverses.

For algebras, $M_3$, $M_4$, $P_3$, $P_4$, $P_5$, and $P_7$, the predicate, `make_projections/4`, was run for $k$ such that $1 \leq k \leq 3$. The runtime data which corresponds with these algebras are recorded in Table 3.2. Again, it is clear that the values of Time/f seem to be converging toward zero, which suggests $T(2, k) = O(f(2, k))$ is an overestimation.
In each case, the number of possible subalgebras, i.e. the number of sublists, is equal to \((2^k - 1)\), for \(k = 1, 2, 3\). That is, 3 possible subalgebras when \(k = 1\), 15 possible subalgebras when \(k = 2\), and 255 possible subalgebras when \(k = 3\). Only two of the six algebras of this table actually reach the worst-cases mentioned above, and these two grow the quickest. It can be seen from the table that the major contributing factor to the actual CPU runtime of `make_projections/4`, for each \(k\), is the number of subalgebras the algebra has. Those algebras that have more subalgebras have runtimes which grow noticeably faster.
Table 3.2: Times to run make_projections/4 on algebras of size $m = 2$, for $1 \leq k \leq 3$.

The data for the rest of the two-element algebras, $M_5$, $P_6$, $P_8$, and $P_9$, appear in Table 3.3. In the case of these four algebras, the predicate is run for the values of $k = 1$, 2, 3, and 4. When $k = 4$, the worst-case number of subalgebras expected is 65535. This number is far greater than the numbers encountered by these algebras, as seen in the table. This would explain the rather quick convergence to zero, demonstrated in all these
Table 3.3: Times to run `make_projections/4` on algebras of size $m = 2$, for $1 \leq k \leq 4$.

The three-element algebras looked at in this thesis are found in Figures 3-5, 3-6, and 3-7. These include all the three-element mono-unary algebras, $M_i$ for $6 \leq i \leq 12$, up to isomorphism, and three chosen three-element bi-unary algebras, $P_1$, $P_{10}$, and $P_{11}$. So, when $m = 3$, the function, $f(m, k)$, becomes

$$f(3, k) = k^3 \cdot 3^k \cdot (2^{3^k}).$$
For all the three-element algebras, with the exception of algebra $M_{12}$, it is possible to run `make_projections/4` for values $k = 1$ and $2$, only. The runtimes, for each $k$, on these algebras are found in Table 3.4, along with the number of subalgebras generated at each level and the results of the complexity test. Again it can be seen, even for just two values of $k$, the consistent rapid growth rate in runtime from $k = 1$ to $k = 2$. Furthermore, the table shows that there is a definite correlation between the number of subalgebras in a run and the time cost of that run, for each $k$. 
<table>
<thead>
<tr>
<th>Algebra</th>
<th>$M_6$</th>
<th></th>
<th>$M_7$</th>
<th></th>
<th>$M_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Time</td>
<td>SA</td>
<td>Time/f</td>
<td>Time</td>
<td>SA</td>
</tr>
<tr>
<td>1</td>
<td>3.8</td>
<td>4</td>
<td>.1583</td>
<td>3.1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>815.4</td>
<td>256</td>
<td>.0221</td>
<td>277.2</td>
<td>75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algebra</th>
<th>$M_9$</th>
<th></th>
<th>$M_{10}$</th>
<th></th>
<th>$M_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Time</td>
<td>SA</td>
<td>Time/f</td>
<td>Time</td>
<td>SA</td>
</tr>
<tr>
<td>1</td>
<td>5.7</td>
<td>7</td>
<td>.2375</td>
<td>3.3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1515.3</td>
<td>511</td>
<td>.0412</td>
<td>145.0</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algebra</th>
<th>$P_1$</th>
<th></th>
<th>$P_{10}$</th>
<th></th>
<th>$P_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Time</td>
<td>SA</td>
<td>Time/f</td>
<td>Time</td>
<td>SA</td>
</tr>
<tr>
<td>1</td>
<td>2.2</td>
<td>1</td>
<td>.0917</td>
<td>4.4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>127.2</td>
<td>9</td>
<td>.0035</td>
<td>278.4</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 3.4: Times to run `make_projections/4` on algebras of size $m = 3$, for $1 \leq k \leq 2$.

![Diagram](image)

Figure 3-7: Algebras $M_{12}$, $P_{10}$, $P_{11}$, and $P_1$.

The algebra, $M_{12}$, is the only three-element algebra for which it is possible to make a run for values of $k$ up to and including 3. Table 3.5 contains the runtime data collected for this algebra. Here it can be seen that the values in the Time/f column converge very...
quickly toward zero, although the runtime for $k = 3$ is comparatively large. In fact, this particular run took 41,800,470 milliseconds, which is approximately eleven and a half hours. This gives further evidence that the size of $k$ has a strong influence on runtime even though the function used for the Big-Oh analysis is a large over-estimate.

<table>
<thead>
<tr>
<th>Algebra</th>
<th>$M_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Time</td>
</tr>
<tr>
<td>1</td>
<td>1.6</td>
</tr>
<tr>
<td>2</td>
<td>73.5</td>
</tr>
<tr>
<td>3</td>
<td>$41800470^*$</td>
</tr>
</tbody>
</table>

Table 3.5: Times to run `make_projections/4` on algebra, $M_{12}$, of size, $m = 3$, for $1 \leq k \leq 3$.

Two four-element algebras are studied in the research, one mono-unary and one bi-unary. Both algebras, $M_1$ and $P_{12}$, are represented in Figure 3-8. The results of the runs done on these algebras are recorded in Table 3.6. The same observations apply to these results, that is, the bigger the $k$, the longer the run, and the more subalgebras for each $k$, the longer the run. In both cases we can clearly see the increase in runtime from $k = 1$ to $k = 2$. Also, in the case of algebra $M_1$ for $k = 2$, the number of subalgebras is substantially larger than for algebra $P_{12}$. This difference is reflected in the disparity between the runtimes of the two.
Figure 3-8: Algebras $M_1$, and $P_{12}$.

<table>
<thead>
<tr>
<th>Algebra</th>
<th>$M_1$</th>
<th>$P_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Time</td>
<td>SA</td>
</tr>
<tr>
<td>1</td>
<td>6.4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>45285.0</td>
<td>7776</td>
</tr>
</tbody>
</table>

Table 3.6: Times to run `make_projections/4` on algebras of size, $m = 4$, for $1 \leq k \leq 2$.

It can also be seen throughout all the tables that the runtimes grow faster from one value of $k$ to another as the size of the algebra, $m$, increases. Table 3.7 contains the runtimes of predicate, `make_projections/4`, averaged over all the algebras of size $m$, for each value of $k$. We see from this table that the times grow rapidly as both $m$ and $k$ increase by increments of one. This can be attributed to the increase in the number of sublists tested, $2^{m^k} - 1$, for each case. This number also grows very rapidly as either $m$ or $k$, or both, increase.
What do these timing results suggest? The fact that \( m \) and \( k \) affect the time so strongly was expected from the complexity analysis. It is obvious that the number of sublists grows rapidly dependent on both \( m \) and \( k \), from the value of \( 2^{mk} - 1 \). What is not so obvious from the complexity analysis is the effect the actual number of subalgebras would have on the runtime. The number of subalgebras that need to be looked at can not be changed, thus their affect on the runtime can not be helped. But, the influence of \( k \) and \( m \) may be lessened, by finding a way of eliminating the testing of a sublist which does not generate a subuniverse. In other words, there may be a way to make the program smarter and thus allow for the predicate to be run on greater values of \( k \) in a reasonable amount of time.

It is necessary to point out why only small values of \( k \) are looked at. Occasionally during a run, the execution error, “aborted: out of address space”, is reported by SICStus 3 Prolog. In the case of running `make.projections/4` on the two-element algebras, \( M_3 \) and \( M_4 \) (Figure 3-2 (a) and (b)), for \( k = 4 \), the execution error is reported after 179,470
and 189,110 milliseconds, respectively. In contrast, for the two-element algebras $M_5$ (Figure 3-2 (c)), $P_6$ (Figure 3-3 (d)), $P_8$, and $P_9$ (Figure 3-4 (b) and (c)), it is possible to run `make_projections/4` for values of $k$ up to and including $k = 4$ (Table 3.3). Similarly, running on the three-element algebra, $P_1$ (Figure 3-7 (d)), for $k = 3$, invokes an execution error after 66,380,390 milliseconds, or approximately 18.44 hours, whereas, the three-element algebra, $M_{12}$ (Figure 3-7 (a)), succeeds in creating the projections for $k = 3$ in approximately 11.50 hours (Table 3.5). The significance of these numbers lies in the number of subalgebras generated and their influence on space costs. Regardless of the time it takes to perform a particular run, the value of $k$ reached is dependent on the number of subalgebras that are generated and stored in the search tree for later use. In theory, this suggests that given the appropriate amount of space, we would be able to run the predicate for any finite $k$ in a finite time. In practice, we have seen from the timing results that in cases of $k$ greater than or equal to 4, the potential runtimes are unrealistic, and trimming these runtimes is unlikely, unless we have apriori knowledge of potential subalgebras. Lessening the space costs is a more realistic goal. Some ideas for reducing space costs are presented in Chapter 4, in the future work section. These include asserting subalgebras as atoms and the use of relational databases for storage of homomorphisms and subalgebras.
3.2 rank/6

In this section, the results of running the predicate, rank/6, are discussed. Recall, rank/6 is used to approximate the rank of an algebraic operation on some unary algebra, $M$. As discussed in Chapter 2, the arguments necessary to this predicate are, $h: B \rightarrow M$, an algebraic operation of $M$, some nonnegative integer, $n$, such that algebra, $B \leq M^n$, and $B \equiv B' \leq M^{n+k}$, for some $k$, a positive fixed integer, and $N$, a bound for the number of homomorphisms to be used to separate $B'$. The remaining two arguments represent a filename for a particular database and the approximated value of the rank of $h$, respectively. There are a total of twenty-four algebras looked at in the research, twelve mono-unary algebras denoted by $M_i$, and twelve bi-unary algebras denoted by $P_i$, where $i = 1, 2, \ldots, 12$. The subalgebras generated for each algebra are denoted by $B_{ij}$, where $i$ is as above and denotes the algebra, and $j$ is a counter on the number of subalgebras generated. For each of twenty-two of the twenty-four algebras, rank/6 is run on all the algebraic operations where $n = 1$, for $N = 1$ and the values of $k$ compatible with the database for the algebra. For the bi-unary algebras $P_1, P_4, P_{10}, P_{11},$ and $P_{12}$, the predicate was run on a few algebraic operations where $n = 2$, the reasoning for this is discussed below. The resulting ranks and runtimes are presented in tables in Appendix A along with the values of $n$, $k$, and $N$ where applicable. The runtimes reported in the tables are in milliseconds (ms). Note that Prolog rounds the times off to ten milliseconds.

---

1Both the one element algebras are ignored because the only algebraic operations of each are the projections, which have rank less than or equal to 0.
so a time of zero means that particular run took less than five milliseconds. Also, the times recorded for running \texttt{rank/6} are based on single runs. Next, a brief discussion of the runtimes is given followed by a look at some of the more interesting results obtained.

The worst-case time complexity of \texttt{rank/6} is

\[
T(k, m, n, N) = O \left( k \cdot \left( \frac{k + n - 1}{n - 1} \right) \cdot 2^{m^{n+k}} \cdot 2^{m^{n+k}} \cdot 2^{m^{n+k}} \cdot N \right),
\]

where \( m \) is the size of the given algebra, \( n \) a fixed nonnegative integer, \( k \) a positive integer, and \( N \) a fixed finite bound. It can be seen that the worst-case time complexity is highly dependent on the number of subalgebras, similar to the worst-case time complexity of predicate, \texttt{make.projections/4}. The relative growth rates in actual runtime for both predicates validate this complexity. With this being the case, an in-depth look at the runtimes of \texttt{rank/6} is not necessary and one may refer to the previous section for a comparable discussion. On the other hand, there are a few particular outcomes that need to be addressed here.

The first point is on a change in efficiency that was made in the midst of doing the rank simulations. The runtime results of the first three algebras, \( M_1, M_3, \) and \( M_4 \), are noticeably less efficient than those of the rest of the algebras due to adjustments made to the code. After \( M_1, M_3, \) and \( M_4 \) were run, some improvements in code efficiency were made, which in turn changed the actual runtime efficiency. For instance, the first clause of the predicate, \texttt{subalgebra/3}, was added, which stops the model from generat-
ing unnecessary subalgebras in the case where the only subalgebra is the algebra itself. Exactly three other such changes were made, which is reflected in the faster runtimes for the later algebras. The important point is that the relative growth rates and the rank results were unaffected.

\begin{figure}[h]
\centering
\begin{tabular}{cc}
(a) M_1 & (b) P_1 \\
\includegraphics[width=1.5in]{M1} & \includegraphics[width=1.5in]{P1}
\end{tabular}
\caption{(a) The algebra M_1 and (b) the algebra P_1.}
\end{figure}

The next two points arise from the runs done on the four-element mono-unary algebra, M_1, shown in Figure 3-9. First of all, it is possible to run \texttt{rank/6} on M_1 for \(n + k = 1\) only, although the database contains projections for \(n + k = 2\). This restriction is due to Prolog address space constraints which are exceeded because algebra, \((M_1)^2\), has 7776 distinct subalgebras each of which are generated a possible 7776 times in the worst-case scenario. When running \texttt{rank/6} on M_1 for \(n + k = 2\), the error message "out of address space" is reported. This error is denoted in the tables by the acronym, OAS. For all the other algebras the runs are bounded only by the previously created database of projections.

Now consider the subalgebras, B_{15} and B_{16}, with universe \(\{0, a\}\) and \(\{0, a, b\}\), respectively, whose results are found in Table 3.8 and Table 3.9. The letters, NA, denote that the rank is not available because it could not be estimated due to Prolog address
space constraints. The runtimes are noticeably shorter when the element \( a \), in algebra \( B_{15} \) or \( B_{16} \), is sent to element \( b \) in algebra \( M_1 \). The difference in these runtimes is further confirmation that the number of subalgebras generated in the search strongly affects the efficiency of the code. In the cases where \( a \) is sent to \( b \), there are fewer subalgebras, \( D \leq M^{n+k} \), for which the homomorphism, \( h' \), lifts to \( D \) because of the restriction that the element, \( c \in M_1 \), satisfies \( f(c) = a \). Since there are fewer subalgebras to be searched, the runtimes are faster for these homomorphisms as opposed to the homomorphisms in which \( a \) is sent to \( 0 \).

<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>( k )</th>
<th>Rank (≤)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{15} \leq M_1 )</td>
<td>( h(0) = h(a) = 0 )</td>
<td>0</td>
<td>1</td>
<td>142680</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = a )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = b )</td>
<td>0</td>
<td>1</td>
<td>70420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
</tbody>
</table>

Table 3.8: The results of running rank/6 on the algebraic operations from algebra, \( B_{15} \), to algebra, \( M_1 \).
<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>k</th>
<th>Rank (≤)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{16} \leq M_1$</td>
<td>$h(0) = h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>73950</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>73990</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>74220</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = a$</td>
<td>0</td>
<td>1</td>
<td>74450</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>73880</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = b$</td>
<td>0</td>
<td>1</td>
<td>35640</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = b, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>35370</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>35480</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
</tbody>
</table>

Table 3.9: The results of running rank/6 on the algebraic operations from algebra, $B_{16}$, to algebra, $M_1$. 

69
The final point to be made on the runtime results of \texttt{rank/6} comes from simulations done on the algebraic operations of the mono-unary algebra, \( M_9 \) (Figure 3-4(c), page 57). The algebra, \( M_9 \), is a three-element mono-unary algebra where \( f(x) = x \), for each \( x \in M_9 \). This algebra has seven subalgebras, one of which is the algebra itself. There are twenty-seven homomorphisms from \( M_9 \) to itself and the results of running \texttt{rank/6} on the first fifteen of these are recorded in tables, Table 3.10 and Table 3.11. When trying to run \texttt{rank/6} on the remaining algebraic operations of \( M_9 \), Prolog reports the error message, “memory allocation failed”. Furthermore, one can see from the tables that the runtimes rapidly increase after the first few homomorphisms have been tested. Recall from the section on complexity analysis in Chapter 2 that the term, \( 2^{m^{n+k}} \), in the complexity of \texttt{rank/6} is a result of the worst-case number of sublists of algebraic operations in the database, for a given algebra. In this particular case, the number of algebraic operations in the database, for \( M_9 \), increases from 1 to 15, when \( k = 0 \) and similarly when \( k = 1 \). Thus, for both \( k = 0 \) and \( k = 1 \), the number of sublists is increasing from \( 2^1 \) to \( 2^{15} \), the worst-case. As a result of this, both the time and space efficiency during these runs is greatly affected. Fortunately, the results for this algebra are theoretically known already \cite{10}, thus the failure to compute results here is not terribly interesting.

\footnote{That is, that mono-unary algebras have rank at most two.}
<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>k</th>
<th>Rank (≤)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{g1} \leq M_9 )</td>
<td>( h(0) = h(a) = h(b) = 0 )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1560</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(a) = 0, h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1570</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(a) = 0, h(b) = b )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1520</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(b) = 0, h(a) = a )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1560</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1560</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = a, h(b) = b )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(b) = 0, h(a) = b )</td>
<td>0</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1580</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = b, h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1610</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = h(b) = b )</td>
<td>0</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1710</td>
</tr>
</tbody>
</table>

Table 3.10: The results of running \( \text{rank}/6 \) on the algebraic operations from algebra, \( B_{g1} \), to algebra, \( M_9 \).
### Table 3.11: The results of running rank/6 on the algebraic operations from algebra, $B_{91}$, to algebra, $M_9$ (continued).

<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>$k$</th>
<th>Rank ($\leq$)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{91} \leq M_9$</td>
<td>$h(0) = a, h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1940</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = a, h(a) = 0$</td>
<td>0</td>
<td>1</td>
<td>1020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2550</td>
</tr>
<tr>
<td></td>
<td>$h(0) = a, h(a) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>2370</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>3900</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = a, h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>5510</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>6900</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>12680</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>14100</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = a, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>29030</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>29890</td>
</tr>
</tbody>
</table>

The rest of this chapter presents the values of rank obtained from the rank/6 trials done on the above mentioned algebras. Before the results are discussed, the order of the steps which the user takes to approximate the rank of an algebraic operation of an algebra are given. Given an algebra, say $M$, there is a database with filename, `algebraM.pl`, that initially contains projections. Given that the database has projections on $M^1$ through $M^l$, consider subalgebras of $M^n$, where $1 \leq n \leq l$, and embeddings, $B \equiv >_\sigma B'$, where
$B' \leq M^{n+k}$ and $n + k \leq l$. For each $B \leq M^n$ ($n \leq l$), and each homomorphism, $h : B \rightarrow M$, chosen by the user, query \texttt{rank/6} with input arguments, $h, n, k, N,$ and \texttt{algebraM.pl}, returning the approximated rank which binds with the sixth argument.

The value of $n$ is dictated by the size of the power of subalgebra, $B$. While $N$, the number of homomorphisms to use for separation of $B'$, is selected by the user. The value of $k$ is incremented by the code, beginning at 0, up to some maximum $k$ such that $n + k \leq l$.

Initially, the number of homomorphisms to use for separation is set at $N = 1$. For $B$, the code then tests every set $Y$ of $N$ homomorphisms from \texttt{algebraM.pl} to determine if they separate $B'$. It then tests if $h$ lifts to $C/Y$, where $B' \leq C \leq M^{n+k}$, and calculates an approximate rank. For a given $N$, if some later homomorphism reports a rank less than any preceding homomorphisms, all the preceding homomorphisms with higher rank must be re-done to ensure that their rank is not any lower. This is because the code selects the homomorphism with the lowest known rank that separates the subalgebra, so there may be a chance that a new homomorphism with lower rank separates the subalgebra being tested. Furthermore, if at any time there is a \texttt{fail} reported in the run, one of two things must be tried. First, retry the homomorphism after all the others are done. If there is still a \texttt{fail}, increase the value of $N$ and try again (i.e. it takes more homomorphisms to separate $B'$), and so on. Continually obtaining \texttt{fail} for $N < n + k$ homomorphisms when $B' \leq M^{n+k}$ suggests that $n + k$ homomorphisms are always required, which forces an infinite rank. It can be said that the logic as applied to these steps is non-monotonic because initial assumptions may be retracted as new information is acquired. The results
for all the runs done are recorded in tables in Appendix A, at the back of the thesis.

Table 3.8 contains the results of running rank/6 on the homomorphisms from algebra, $B_{15}$, a subalgebra of algebra, $M_1$, to $M_1$, for $k = 0$ and $k = 1$. The algebra, $B_{15}$, is the two-element subalgebra of $M_1$ containing the elements $\{0, a\}$. Notice from the table that the third homomorphism defined by, $h(0) = 0$ and $h(a) = b$, has a reported rank of less than or equal to one, for $k = 0$. From Example 4, in Chapter 1 (15), it is known that this homomorphisms has rank less than or equal to two. The disparity between the two results is due to the restriction on $k$ for this particular algebra. Because $k$ can not exceed zero and $N$ is at least one, $k$ is never greater than or equal to $N$, which is a necessary condition for there to be a fail lifting to $M^{k+1}\{\hat{c}\}$. The point is that a result for $k = 1$ cannot be computed for this algebra and thus a true comparison of the results cannot be made. The only way to remedy this problem is to make the code more efficient. Some thoughts on this are presented in the next chapter.

For the remaining mono-unary algebras, all the runs were done with $n = 1$ and all reported computed ranks were less than or equal to one. This was to be expected for it is known that all mono-unary algebras are rank at most two [10], and furthermore, that all mono-unary algebras of size three or less have rank of at most one, which follows from work done in [10].

The next algebra of interest is the three-element bi-unary algebra, $P_1$, shown in Figure 3-9. That this algebra has an infinite rank is shown implicitly in [11]. The algebra, $P_1$, has one subalgebra, itself, and $(P_1)^2$ (Figure 3-10) has nine subalgebras, one
of them being itself. The results from running \( \text{rank/6} \) on these two subalgebras, denoted \( B_{11} \) and \( B_{12} \), respectively, are shown in Table 3.12. For \( B_{11} \), there is one homomorphism, the projection, from \( B_{11} \) to \( P_1 \). So, for both \( k = 0 \) and \( k = 1 \), with \( n = 1 \) and \( N = 1 \), the reported rank is zero. On the other hand, for \( B_{12} \) there exists four algebraic operations of \( P_1 \), the two projections, the minimum of both coordinates and the maximum of the coordinates. In the case of running these four homomorphisms, \( n = 2 \), \( k = 0 \), are fixed and \( N = 1 \) initially. As expected, the two projections produced rank zero. What is of interest here is that the other two homomorphisms, defined by \( h(x, y) = \min(x, y) \) and \( h(x, y) = \max(x, y) \) produced a failure in the query when \( N = 1 \) and that rank is less than or equal to one when \( N = 2 \). These results confirm the expected results because \( \text{rank}(P_1) = \infty \). What is happening here is that when \( n = 2 \), it takes two projections to separate \( B_{12} \), thus when \( N = 1 \) separation will not happen. Once \( N \) is set to two, then the two projections necessary for separation are available and the rank is reported as one. Also, since neither of the two homomorphisms is of rank one when \( N = 1 \), there are no other choices for separation but projections. The idea is that as \( n + k \) increases, so must \( N \), which means the rank is infinite.

\[
\begin{array}{c}
00 \\
b0 \\
a0 \\
b \quad \quad \quad \quad a \\
bb \quad \quad \quad \quad ab \\
0b \\
\end{array}
\]

Figure 3-10: The algebra, \( B_{12} \), a subalgebra of \( (P_1)^2 \).
The bi-unary algebra, $P_1$, mentioned above is referred to as a \textit{chain} algebra because the two functions of the algebra are similar but they work in the opposite direction, seen in Figure 3-9 (page 67). This algebra is the only bi-unary algebra whose rank we know anything about. The rest of the bi-unary algebras presented in this chapter are those for which \textit{rank/6 fails} for at least one homomorphism. Now consider the four-element bi-unary chain, $P_{12}$, of Figure 3-8(b) (page 62) and the two subalgebras, $B_{121} = P_{12}$ and $B_{122} = (P_{12})^2$. The results of running \textit{rank/6} on the homomorphisms from these two subalgebras to $P_{12}$ are presented in Table 3.13. Notice that the algebraic operations from $B_{121}$ and $B_{122}$ are similar to those of algebra $P_1$ and the rank results of running these algebraic operations are also similar to the results of $P_1$ (Table 3.12). Specifically,
the homomorphisms, \( h(x, y) = \min(x, y) \) and \( h(x, y) = \max(x, y) \), both fail the rank query when \( N = 1 \), and return a rank of one when \( N = 2 \). Furthermore, although \((P_1)^2\) has nine subalgebras and \((P_{12})^2\) has sixteen subalgebras, all the subalgebras take on the same patterns. These similarities between the two algebras suggest that the rank of \( P_{12} \) is infinite, equal to that of \( P_1 \). That is, \( N \) must be greater than or equal to \( n + k \) in order for there to be enough homomorphisms to separate \((P_{12})^{n+k}\).

<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>( k )</th>
<th>( N )</th>
<th>Rank ((\leq))</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{121} = P_{12} )</td>
<td>( h(x) = x )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>( B_{122} = (P_{12})^2 )</td>
<td>( h(x, y) = \min(x, y) )</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>15150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>15050</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = x )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = y )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = \max(x, y) )</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>15070</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>15060</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.13: The results of running rank/6 on the algebraic operations from algebras, \( B_{121} \) and \( B_{122} \), to algebra, \( P_{12} \).
<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>k</th>
<th>N</th>
<th>Rank (≤)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{41} = P_4$</td>
<td>$h(0) = 0$, $h(1) = 1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{42} = (P_4)^2$</td>
<td>$h(x, y) = \min (x, y)$</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>fail</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h(x, y) = x$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h(x, y) = y$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(x, y) = \max (x, y)$</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>fail</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>660</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.14: The results of running rank/6 on the algebraic operations from algebras, $B_{41}$ and $B_{42}$, to algebra, $P_4$.

To get a look at the behavior of the previous two chain algebras when $n + k = 3$, it is wise to look at the two-element chain algebra, $P_4$ (Figure 3-3, page 56), for which the computational model has the capability of reaching $n + k = 3$. Again, $P_4$ has similar...
subalgebras and homomorphisms as both $P_1$ and $P_{12}$, as well as the same rank results for $n + k = 1$ and 2 (Table 3.14). Together these facts lead to the belief that $P_4$ also has an infinite rank. In the case of $P_4$, it is possible to look at the algebraic operations corresponding to $(P_4)^3$. The algebra $(P_4)^3$ has sixty-four homomorphism from it to $P_4$, each of which fails a rank query when $N = 1$ or 2, and returns a rank of one when $N = 3$.

The last two algebras to be presented are the three-element bi-unary algebras, $P_{10}$ and $P_{11}$ (Figure 3-7, page 60), which are very close in structure to algebra $P_1$. The algebra, $P_{11}$, has two subalgebras, the algebra itself, denoted $B_{111}$, and the one-element algebra with universe $\{0\}$, denoted $B_{112}$. Furthermore, the algebra $(P_{11})^2$ has thirteen subalgebras with $(P_{11})^2$ denoted by $B_{113}$. The results of running rank/6 with these three subalgebras and the corresponding algebraic operations appear in Table 3.15. For this algebra, there is one more algebraic operation from $(P_{11})^2$ to $P_{11}$, than there was from $(P_1)^2$ to $P_1$, and it also fails the rank query when $N = 1$. Again, this leads to the belief that this algebra has infinite rank as well, but it is possible that this one extra homomorphism translates to an algebraic operation with a finite rank for larger $k$.

---

3The results of running $(P_4)^3$ are not presented in table format because of the large number of homomorphisms.
<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>k</th>
<th>N</th>
<th>Rank (≤)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{B}<em>{111} = \mathbb{P}</em>{11} )</td>
<td>( h(x) = 0 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>( h(x) = x )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \mathbb{B}<em>{112} \leq \mathbb{P}</em>{11} )</td>
<td>( h(0) = 0 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \mathbb{B}<em>{113} = (\mathbb{P}</em>{11})^2 )</td>
<td>( h(x, y) = 0 )</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = \min(x, y) )</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = x )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = y )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(x, y) = \max(x, y) )</td>
<td>0</td>
<td>1</td>
<td>fail</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.15: The results of running \( \text{rank}/6 \) on the algebraic operations from algebras, \( \mathbb{B}_{111}, \mathbb{B}_{112} \) and \( \mathbb{B}_{113} \), to algebra, \( \mathbb{P}_{11} \).

The results for algebra, \( \mathbb{P}_{10} \), are seemingly similar to the rest but are less conclusive since fewer runs were done and there is no prior theoretical knowledge of its rank. This algebra has three subalgebras, \( \mathbb{B}_{101}, \mathbb{B}_{102}, \) and \( \mathbb{B}_{103} \), which generate a total of five algebraic operations, two from \( \mathbb{B}_{101} \), one from \( \mathbb{B}_{102} \), and two from \( \mathbb{B}_{103} \), to \( \mathbb{P}_{10} \). The results
of running rank/6 for these operations are given in Table 3.16. As usual, the ranks of the algebraic operations when \( n = 1 \) are either zero or one. The power algebra, \((P_{10})^2\), has eighteen homomorphisms which map elements from it to \(P_{10}\), all of which, aside from the two projections, failed the rank query when \( N = 1 \). Again as we have no a priori knowledge, these results are far less suggestive of infinite rank. Like all the rest, however, it suggests that this algebra may be worth looking at theoretically to see if it has an infinite rank.

<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>( k )</th>
<th>Rank (( \leq ))</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{101} = P_{10} )</td>
<td>( h(x) = 0 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>( h(x) = x )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( B_{102} \leq P_{10} )</td>
<td>( h(0) = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( B_{103} \leq P_{10} )</td>
<td>( h(x) = 0 )</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td></td>
<td>( h(x) = x )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.16: The results of running rank/6 on the algebraic operations from algebras, \( B_{101}, B_{102} \) and \( B_{103} \), to algebra, \( P_{10} \).
The results of the runs done on the remaining bi-unary algebras can be found in Appendix A along with the rest of the rank simulation results. For all the algebras, aside from the bi-unary algebras mentioned above, $n = 1$ is the only case looked at. Further study for higher values of $n$ and $k$ on all the algebras presented, may produce different results.

3.3 Summary

The results of running both of the predicates, `make_projections/4` and `rank/6`, are found in this chapter or Appendix A. The actual runtimes of `make_projections/4` are compared to the results of the complexity analysis, given in detail in Chapter 2. A discussion of simulations done using `rank/6` is presented on both the runtimes of the simulations and the rank approximations obtained. The focus of this discussion is centered around the six more interesting algebras, $M_1$, $P_1$, $P_4$, $P_{10}$, $P_{11}$, and $P_{12}$, of the twenty-four studied. It is shown that the computational model provides simulated results consistent with known theoretical results of rank. Overall, the system is shown to work on the algebras introduced and the process of approximating rank with the computational model is put to practice. It is important to emphasize, that in approximating rank, we are approximating an upper bound.
Chapter 4

Conclusion

4.0 Introduction

The conclusions of the thesis are presented in the first section of Chapter 4. The second section of this chapter presents various ideas on possible future research in the area of the thesis.

4.1 Conclusion

The research of this thesis presents a computational model for the purpose of finding a practical connection between the fields of Mathematics and Computer Science through the application of Natural Language Processing and Logic Programming techniques to the study of algebras. Through the process of exploring this connection, the author has created a computational model in the programming language Prolog, which can be
used to approximate the rank of unary algebras and their algebraic operations. The computational model was structured after the ideas of Natural Language Processing, by viewing universal algebra as a language. By creating this model and obtaining feasible results, the major goal of the research has been met.

The results of the research include tools for working with finite unary algebras. The centerpiece of these tools is the predicate which is used to approximate the rank of algebras and their algebraic operations. Although definitive answers can not be calculated using the predicate for rank, much can be learned in its finite approximations. The rank predicate is used for three purposes. The first purpose is to confirm computationally, what has been proven theoretically. The second purpose is to spot trends or tendencies in the $Kth$ approximation ranks of algebras. The third purpose of the rank predicate is to find algebras that show potentially interesting attributes for further theoretical work, thereby avoiding fruitless work on uninteresting algebras.

The rest of the tools created to facilitate the implementation of the rank predicate are useful in their own right. These tools include Prolog predicates which generate subalgebras, homomorphisms and factor algebras on finite unary algebras. One Prolog predicate partitions an integer into a finite list and another builds a finite power algebra of a unary algebra. Each of these tools as well as a few others can be used outside the scope of the rank predicate for which they were written.

The biggest obstacle to the research done in this thesis is the computing limitations of the computational model. As is stated in Chapter 3, the approximation of rank is
bounded to small values of $k$ in most instances. Other than the work done on the trivial unary algebras, the highest value of $k$ reached while approximating the rank of an algebraic operation is four. It was shown that this limit is due to the time and space used while generating all the possible subalgebras of an algebra. Generating these subalgebras is a necessary step in the computational model. Each rank approximation computation continues until the finite time and space limitations are encountered. The author has come up with some possible remedies for this problem which will be discussed in the next section on possible future work which may be done on the research in question.

4.2 Future Work

This section is a discussion of possible future work to be done on the computational model. First, a presentation of some ideas which may help to cut the time and space costs incurred while running the current model. This is followed with suggestion for expanding the current work. In each case, the discussion is limited to a brief overview of the idea in mind.

One of the more obvious ideas is to assert subalgebras as atoms once they are found. As the code stands, a subalgebra may be generated repeatedly throughout the course of a run. Even though generating all the subalgebras of an algebra once can be quite expensive, and this would not change, it would make sense to not generate them again. In conjunction with this idea, another idea would be to save the subalgebras of an algebra as a lattice structure. In some cases, a goal of the computational model is to
find all the subalgebras that lie between two specific subalgebras. By viewing all the subalgebras of an algebra as a lattice, finding all the required subalgebras between two subalgebras becomes a matter of finding the endpoints of some subinterval in the lattice. The subinterval can then be traversed as needed.

Another idea which is worth exploring comes from Ceri, Gottlob, and Tanca's, Logic Programming and Databases, [4], in which they discuss the use of relational databases in conjunction with Prolog. The idea is to create an interface between Prolog and an external relational database for dealing with large data sets. Upon further research, it has been found that the SICStus version of Prolog, which is used for this thesis, has a built-in interface which does exactly this. Once again, the biggest problem of this thesis is the generation and storage of potentially large quantities of subalgebras. Also, the databases of homomorphisms of known rank is constantly growing. It is these two things which account for most of the space consumption while running the rank predicate.

Outside trying to improve the efficiency of the current code, the groundwork has been laid for expansion into other types of algebras and tools of algebras. The ultimate goal would be to create a Prolog system which could be used as an algebra calculator. As a final note, UNBC is in the process of receiving a new super computer that may increase the capacity of the code as it stands.
4.3 Summary

The research of this thesis provides an original and unique computational model for the purposes of approximating rank. The model could prove a valuable tool for both algebraists and topologists when used for duality theory. Furthermore, the computational model created in the research contains many secondary tools that can be used by algebraists.
Bibliography


Appendix A

Appendix: Tables

Appendix A contains the tables of results obtained from running rank/6 on some mono-unary and bi-unary algebras. The tables include the subalgebra names, the homomorphisms, the size of \( k \), the approximated rank, and the runtimes in milliseconds.

Some tables include a listing of the value of \( N \), for those without this listing \( N = 1 \).
<table>
<thead>
<tr>
<th>Subalgebra</th>
<th>Homomorphism</th>
<th>k</th>
<th>Rank ($\leq$)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{11} = M_1$</td>
<td>$h(0) = h(a) = h(b) = h(c) = 0$</td>
<td>0</td>
<td>1</td>
<td>35700</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = h(b) = 0, h(c) = a$</td>
<td>0</td>
<td>1</td>
<td>35460</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = h(b) = 0, h(c) = b$</td>
<td>0</td>
<td>1</td>
<td>35760</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = h(c) = 0, h(a) = a$</td>
<td>0</td>
<td>1</td>
<td>35530</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = h(c) = a$</td>
<td>0</td>
<td>1</td>
<td>35510</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = a, h(c) = b$</td>
<td>0</td>
<td>1</td>
<td>35470</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = h(c) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>35790</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = b, h(c) = a$</td>
<td>0</td>
<td>1</td>
<td>35710</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = h(c) = b$</td>
<td>0</td>
<td>1</td>
<td>35850</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{11} = M_1$</td>
<td>$h(0) = h(b) = 0, h(a) = a, h(c) = c$</td>
<td>0</td>
<td>1</td>
<td>36380</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = a, h(c) = c$</td>
<td>0</td>
<td>1</td>
<td>37060</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(x) = x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{12} \leq M_1$, $B_{12} = {0}$</td>
<td>$h(0) = 0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{13} \leq M_1$, $B_{13} = {0, b}$</td>
<td>$h(0) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>108100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>108250</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{14} \leq M_1$, $B_{14} = {0, a, c}$</td>
<td>$h(0) = h(a) = h(c) = 0$</td>
<td>0</td>
<td>1</td>
<td>73870</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0$, $h(c) = a$</td>
<td>0</td>
<td>1</td>
<td>73660</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0$, $h(c) = b$</td>
<td>0</td>
<td>1</td>
<td>73840</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0$, $h(a) = a$, $h(c) = c$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{15} \leq M_1$, $B_{15} = {0, a}$</td>
<td>$h(0) = h(a) = 0$</td>
<td>0</td>
<td>1</td>
<td>142680</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0$, $h(a) = a$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0$, $h(a) = b$</td>
<td>0</td>
<td>1</td>
<td>70420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank ($\leq$)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{16} \leq M_1$</td>
<td>$h(0) = h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>73950</td>
</tr>
<tr>
<td>$B_{16} = {0, a, b}$</td>
<td>$h(0) = h(a) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>73990</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>74220</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = a$</td>
<td>0</td>
<td>1</td>
<td>74450</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>73880</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = b$</td>
<td>0</td>
<td>1</td>
<td>35640</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = b, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>35370</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>35480</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>NA</td>
<td>OAS</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(B_{31} = M_3)</td>
<td>(h(0) = h(1) = 0)</td>
<td>0</td>
<td>1</td>
<td>470</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2310</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>31960</td>
</tr>
<tr>
<td></td>
<td>(h(0) = 0, h(1) = 1)</td>
<td>0</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(B_{32} \leq M_3)</td>
<td>(h(0) = 0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(B_{32} = {0})</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>( B_{41} = M_4 )</td>
<td>( h(0) = h(1) = 0 )</td>
<td>0</td>
<td>1</td>
<td>850</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>4150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>58360</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, \ h(1) = 1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 1, \ h(1) = 0 )</td>
<td>0</td>
<td>1</td>
<td>860</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>4160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>58160</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(1) = 1 )</td>
<td>0</td>
<td>1</td>
<td>860</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>4190</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>58680</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>---</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>(B_{42} \leq M_4,)</td>
<td>(h(1) = 0)</td>
<td>0</td>
<td>1</td>
<td>1700</td>
</tr>
<tr>
<td>(B_{42} = {1})</td>
<td></td>
<td>1</td>
<td>1</td>
<td>8330</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>116990</td>
</tr>
<tr>
<td>(h(1) = 1)</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(B_{43} \leq M_4,)</td>
<td>(h(0) = 0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(B_{43} = {0})</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(h(0) = 1)</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1680</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>8400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>117300</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{51} = M_5$</td>
<td>$h(0) = 0$, $h(1) = 1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 1$, $h(1) = 0$</td>
<td>0</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>35650</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>( B_{61} = M_6 )</td>
<td>( h(0) = h(a) = h(b) = 0 )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1040</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(a) = 0, h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1060</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(a) = 0, h(b) = b )</td>
<td>0</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1040</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(b) = 0, h(a) = a )</td>
<td>0</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1040</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1050</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = a, h(b) = b )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(0) = h(b) = 0, h(a) = b )</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1080</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = b, h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1110</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, h(a) = h(b) = b )</td>
<td>0</td>
<td>1</td>
<td>190</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1220</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{62} \leq M_6$, $B_{62} = {0, b}$</td>
<td>$h(0) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>430</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>430</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2590</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$B_{63} \leq M_6$, $B_{63} = {0}$</td>
<td>$h(0) = 0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{64} \leq M_6$, $B_{64} = {0, a}$</td>
<td>$h(0) = h(a) = 0$</td>
<td>0</td>
<td>1</td>
<td>430</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = b$</td>
<td>0</td>
<td>1</td>
<td>430</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2530</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{71} = M_7$</td>
<td>$h(0) = h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{72} \leq M_7$</td>
<td>$h(0) = 0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{72} = {0}$</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_{73} \leq M_7$</td>
<td>$h(0) = h(a) = 0$</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$B_{73} = {0, a}$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{s1} = M_s$</td>
<td>$h(0) = h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = a$</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = b, h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>510</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>520</td>
</tr>
<tr>
<td>$B_{s2} \leq M_{s_2}$, $B_{s2} = {b}$</td>
<td>$h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1140</td>
</tr>
<tr>
<td></td>
<td>$h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{83} \leq M_8$, $B_{83} = {0, b}$</td>
<td>$h(0) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0$, $h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = b$, $h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1070</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1050</td>
</tr>
<tr>
<td>$B_{84} \leq M_8$, $B_{84} = {0}$</td>
<td>$h(0) = 0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = b$</td>
<td>0</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2130</td>
</tr>
<tr>
<td>$B_{85} \leq M_8$, $B_{85} = {0, a}$</td>
<td>$h(0) = h(a) = 0$</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>960</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0$, $h(a) = a$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = b$</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>980</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank ($\leq$)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{9!} \leq M_9$</td>
<td>$h(0) = h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1560</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1570</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1520</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = a$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1560</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1560</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = 0, h(a) = b$</td>
<td>0</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1580</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = b, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1610</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1710</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_9 \leq M_9$</td>
<td>$h(0) = a, h(a) = h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1940</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(b) = a, h(a) = 0$</td>
<td>0</td>
<td>1</td>
<td>1020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2550</td>
</tr>
<tr>
<td></td>
<td>$h(0) = a, h(a) = 0, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>2370</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>3900</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = a, h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>5510</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>6900</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>12680</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>14100</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(a) = a, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>29030</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>29890</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{101} = M_{10}$</td>
<td>( h(0) = h(a) = h(b) = 0 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, \ h(a) = a, \ h(b) = b )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, \ h(a) = b, \ h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td>$B_{102} \leq M_{10}$, ( B_{102} = {a, b} )</td>
<td>( h(a) = h(b) = 0 )</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>( h(a) = a, \ h(b) = b )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( h(a) = b, \ h(b) = a )</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td>$B_{103} \leq M_{10}$, ( B_{103} = {0} )</td>
<td>( h(0) = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>$k$</td>
<td>Rank ($\leq$)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-----</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>$B_{111} = M_{11}$</td>
<td>$h(0) = 0, h(a) = h(b) = a$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1$</td>
<td>$1$</td>
<td>$200$</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(a) = h(b) = b$</td>
<td>$0$</td>
<td>$0$</td>
<td>$6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td></td>
<td>$h(0) = a, h(a) = h(b) = 0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$16$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1$</td>
<td>$1$</td>
<td>$220$</td>
</tr>
<tr>
<td>$B_{112} \leq M_{11}$, $B_{102} = {a, b}$</td>
<td>$h(0) = 0, h(a) = a$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td></td>
<td>$h(0) = a, h(a) = 0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$10$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1$</td>
<td>$1$</td>
<td>$400$</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>N</td>
<td>Rank (≤)</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------</td>
<td>---</td>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>$B_{121} = M_{12}$</td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = a, h(a) = b, h(b) = 0$</td>
<td>0</td>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>72045720</td>
</tr>
<tr>
<td></td>
<td>$h(0) = b, h(a) = 0, h(b) = a$</td>
<td>0</td>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>370</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>72013680</td>
</tr>
<tr>
<td>$B_{11} = P_1$</td>
<td>$h(0) = 0, h(a) = a, h(b) = b$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$B_{12} = (P_1)^2$</td>
<td>$h(x,y) = \min(x,y)$</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>$h(x,y) = x$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(x,y) = y$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(x,y) = \max(x,y)$</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>----</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{31} = P_3$</td>
<td>$h(0) = h(1) = 0$</td>
<td>0</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(1) = 1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$B_{32} \leq P_3$; $B_{32} = {0}$</td>
<td>$h(0) = 0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>N</td>
<td>Rank (≤)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>$B_{41} = P_4$</td>
<td>$h(0) = 0, h(1) = 1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$B_{42} = (P_4)^2$</td>
<td>$h(x, y) = \min(x, y)$</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>620</td>
</tr>
<tr>
<td></td>
<td>$h(x, y) = x$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(x, y) = y$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(x, y) = \max(x, y)$</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>660</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>( \text{Rank} ) (( \leq ))</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>( B_{51} = P_5 )</td>
<td>( h(0) = h(1) = 0 )</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>1240</td>
</tr>
<tr>
<td></td>
<td>( h(0) = 0, \ h(1) = 1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( B_{52} \leq P_5 )</td>
<td>( h(0) = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( B_{52} = {0} )</td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( B_{61} = P_6 )</td>
<td>( h(0) = 0, \ h(1) = 1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{71} = P_7$</td>
<td>$h(0) = h(1) = 0$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>1740</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 0, h(1) = 1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 1, h(1) = 0$</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>1760</td>
</tr>
<tr>
<td></td>
<td>$h(0) = h(1) = 1$</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>1780</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(B_{72} \leq P_7,)</td>
<td>(h(1) = 0)</td>
<td>0</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>(B_{72} = {1})</td>
<td>1</td>
<td>170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3460</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(h(1) = 1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B_{73} \leq P_7,)</td>
<td>(h(0) = 0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(B_{73} = {0})</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(h(0) = 1)</td>
<td>0</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$B_{81} = P_8$</td>
<td>$h(0) = 0, h(1) = 1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 1, h(1) = 0$</td>
<td>0</td>
<td>1</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>540</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>63210</td>
</tr>
<tr>
<td>$B_{91} = P_9$</td>
<td>$h(0) = 0, h(1) = 1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$h(0) = 1, h(1) = 0$</td>
<td>0</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>61060</td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>Rank (≤)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>$$B_{101} = P_{10}$$</td>
<td>$$h(x) = 0$$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$$B_{101} = {0, a, b}$$</td>
<td>$$h(x) = x$$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$$h(x) = x$$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$$B_{102} \leq P_{10}$$, $$B_{102} = {0}$$</td>
<td>$$h(0) = 0$$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$$B_{103} \leq P_{10}$$, $$B_{163} = {0, a}$$</td>
<td>$$h(x) = 0$$</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$$h(x) = x$$</td>
<td>1</td>
<td>1</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>$$h(x) = x$$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>N</td>
<td>Rank (≤)</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------</td>
<td>---</td>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>B_{111} = P_{11}</td>
<td>h(x) = 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>h(x) = x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B_{112} ≤ P_{11}</td>
<td>h(0) = 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B_{113} = (P_{11})^2</td>
<td>h(x, y) = 0</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>h(x, y) = min(x, y)</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>h(x, y) = x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>h(x, y) = y</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>h(x, y) = max(x, y)</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Subalgebra</td>
<td>Homomorphism</td>
<td>k</td>
<td>N</td>
<td>Rank (≤)</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>---</td>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>B_{121} = P_{12}</td>
<td>h(x) = x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B_{122} = (P_{12})^2</td>
<td>h(x, y) = \min(x, y)</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>h(x, y) = x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>h(x, y) = y</td>
<td>0</td>
<td>i</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>h(x, y) = \max(x, y)</td>
<td>0</td>
<td>1</td>
<td>fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

Appendix: Code

The appendix contains the predicates written and implemented for the purpose of approximating the rank of homomorphisms of finite unary algebras. All code is written using SICStus Prolog. The appendix has been broken into the files that the code is contained in. Each file includes a header containing the name of the file, a brief description of the file, the author's name, creation and modification dates, the names of the predicates in the file, and the names of the calling predicates where applicable. Each predicate begins with detailed documentation. Following the convention of the SICStus Prolog manual [12], when introducing predicates, the arguments of the predicates have one of the following forms:

:ArgName - This argument should be instantiated to a term denoting a goal or a clause or a predicate name, or which otherwise needs special handling of module prefixes.

+ArgName - This argument should be instantiated to a non-variable term.
-ArgName - This argument should be uninstantiated.

?ArgName - This argument may or may not be instantiated.

All code was written and implemented by the author of this thesis, Richard K. Little, during the years of 1998 and 1999.
FILE: rank.pl

Approximates the rank of an algebraic operation
of a fixed finite unary algebra.

BY: Rich Little
CREATED: 16 Dec 1998
MODIFIED: 23 Jan 1999 (changed variables)
01 Feb 1999 (added for_each_part/9)
03 Mar 1999 (moved the call to get_Ys/5)

PREDICATES: rank/6
rank/9
for_each_k/9
for_each_part/9
for_each_D/8
for_each_C/7
there_exists_Y/7
for_each_g/3

CALLED-BY:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

% rank(+HOMO,+SMALLN,+K,+N,+FILE,-RANK)
% The argument, HOMO, is instantiated with a clause form,
% homomorphism/5, an algebraic operation of a given algebra. The
% rank of the algebraic operation is approximated for values up
% to some K, and binds with argument, RANK. The argument, FILE,
% is the filename of a file containing the database of algebraic
% operations with known rank.

rank(homomorphism(R,K,algebra(B,FB),algebra(M,FM),[DH,H]),Sn,K,N,F,R) :-
  list_to_ord_set(B,OB),
  ord_functions(FB,OFB),
  list_to_ord_set(M,OM),
  ord_functions(FM,OFM),
  list_to_ord_set(DH,ODH),
  list_to_ord_set(H,OH),
  is_algebra(algebra(OB,OFB)),
  is_algebra(algebra(OM,OFM)),
  is_homomorphism(homomorphism(R,K,algebra(OB,OFB),algebra(OM,OFM),[ODH,O]),
  make_M_to_the_n(algebra(OM,OFM),Sn,algebra(Mn,FMn)),
  is_algebra(algebra(Mn,FMn)),
  list_to_ord_set(Mn,OMn),
  ord_functions(FMn,OFMn),
  subalgebra(algebra(OB,OFB),algebra(OM,OFM),Sn,K,N,F,-2,R).

% rank(+HOMO,+ALG_B,+ALG_M,+SMALLN,+K,+N,+FILE,+MAXRANK,-RANK)
% The argument, HOMO, represents a homomorphism from an algebra,
ALG_B, to an algebra, ALG_M, for some positive integer K. If
the rank of HOMO already exists in the database, FILE, return
the rank, RANK. Else, the rank is approximated and the argument
RANK binds with the maximum rank, MAXRANK, plus one. In this case
the database of homomorphisms is consulted as facts. Sometimes it
is more prudent to leave the database as a file and read the
terms in one at a time. The lines of clause one that our commented
are used for this purpose.

```
rank(Homo,Alg_B,Alg_M,Sn,K,N,F,MR,R) :-
    open(F,read,Stream),
    read(Stream,Term),
    check_dBase(Term,Stream,K,Homo,Alg_B,Alg_M,R),
    close(Stream), !.
```

```
homomorphism(R,K,Alg_B,Alg_M,Homo), !.
```

```
rank(Homo,Alg_B,Alg_M,Sn,K,N,F,MR,R) :-
    !, for_each_k(0,Homo,Alg_B,Alg_M,Sn,K,N,F,MR,TR),
    R is TR + 1,
    open(F,append,Stream),
    write(Stream,homomorphism(R,K,Alg_B,Alg_M,Homo)),
    format(Stream, "-w", []),
    format(Stream, "-n", []),
    close(Stream),
    assert((homomorphism(R,K,Alg_B,Alg_M,Homo))), !.
```

```
for_each_k(I,Homo,Alg_B,Alg_M,Sn,K,N,R,R) :-
    I <= K,
    SnPlusK is Sn + I,
    make_M_to_the_n(Alg_M,SnPlusK,algebra(MnK,FMnK)),
    list_to_ord_set(MnK,OMnK),
    ord_functions(FMnK,OFMnK),
    is_algebra(algebra(OMnK,OFMnK)),
    partition_k(I,Sn,PList),
    !, for_each_part(PList,Homo,Alg_B,Alg_M,algebra(OMnK,OFMnK),K,N,MR,TR),
    I is I + 1,
```

```
for_each_part(+PARTS,+HOMO,+ALG_B,+ALG_M,+ALG_MnK,+K,+N,+MAXR,-R);
    If all partitions in the list argument, PARTS, are exhausted,
    then argument, R, binds with MAXR, the maximum rank of
    homomorphism, HOMO. Otherwise, recursively traverse list, PARTS,
    to approximate the rank of homomorphism, HOMO.
```
for_each_part([Part|Parts], Homo, Alg_B, Alg_M, Alg_MnK, K, N, MR, R) :-
  embed_B(Alg_B, Part, algebra(Bprm, FBprm)),
  list_to_ord_set(Bprm, OBprm),
  ord_functions(FBprm, OFBprm),
  is_algebra(algebra(OBprm, OFBprm), Alg_MnK),
  make_hprime(Homo, algebra(OBprm, OFBprm), Hprm),
  is_homomorphism(homomorphism(R, K, algebra(OBprm, OFBprm), Alg_M, Hprm)),
  bagof(Alg_D, subalgebra(algebra(OBprm, OFBprm), Alg_M, Alg_MnK, SubAlgs_D),
  !, for_each_D(SubAlgs_D, Hprm, Alg_M, Alg_Bprm, K, N, MR, TR),

for_each_D([algebra(D, FD) | Alg_Ds], Hprm, Alg_M, Alg_Bprm, K, N, MR, R) :-
  list_to_ord_set(D, OD),
  ord_functions(FD, OFD),
  lifts(Hprm, algebra(OD, OFD), Alg_M),
  get_Ys(Alg_M, algebra(OD, OFD), K, N, Ys),
  bagof(Alg_C, subalgebra(Alg_Bprm, Alg_M, Alg_C, algebra(OD, OFD)), SubAlgs_C),
  !, for_each_C(SubAlgs_C, Hprm, Alg_M, Alg_Bprm, Ys, MR, TR),
  for_each_D(Alg_Ds, Hprm, Alg_M, Alg_Bprm, K, N, TR, R).
for_each_D([Alg_D | Alg_Ds], Hprm, Alg_M, Alg_Bprm, K, N, MR, R) :-
  for_each_D(Alg_Ds, Hprm, Alg_M, Alg_Bprm, K, N, MR, R).

for_each_C([], Hprm, Alg_M, Alg_Bprm, Ys, R, R).
for_each_C([algebra(C, FC) | Alg_Cs], Hprm, Alg_M, Alg_Bprm, Ys, MR, R) :-
  list_to_ord_set(C, OC),
  ord_functions(FC, OFC),
  !, there_exists_Y(Ys, algebra(OC, OFC), Alg_M, Alg_Bprm, Hprm, MR, TR),
  for_each_C(Alg_Cs, Hprm, Alg_M, Alg_Bprm, Ys, TR, R).

there_exists_Y([], Hprm, Alg_M, Alg_Bprm, Ys, R, R).
there_exists_Y([algebra(C, FC) | Alg_Cs], Hprm, Alg_M, Alg_Bprm, Ys, MR, R) :-
  list_to_ord_set(C, OC),
  ord_functions(FC, OFC),
  !, there_exists_Y(Ys, algebra(OC, OFC), Alg_M, Alg_Bprm, Hprm, MR, TR),
  for_each_C(Alg_Cs, Hprm, Alg_M, Alg_Bprm, Ys, TR, R).
a fail of the whole predicate by the first clause.

there_exists_Y([Y|Ys],Alg_C,Alg_M,Alg_Bprm,Hprm,R,NewR) :-
    separates(Y,Alg_Bprm,Fact_Alg),
    factor_algebra(Alg_C,Y,Alg_Bprm,Fact_Alg,algebra(CmodY,FCmodY)),
    list_to_ord_set(CmodY,OCmodY),
    ord_functions(FCmodY,OFCmodY),
    is_algebra(algebra(OCmodY,OFCmodY)),
    lifts(Hprm,algebra(OCmodY,OFCmodY),Alg_M),
    for_each_g(Y,R,NewR).
there_exists_Y([Y|Ys],Alg_C,Alg_M,Alg_Bprm,Hprm,R,NewR) :-

% for_each_g(+Y,+TEMPRANK,-MAXRANK)
% Argument. TEMPRANK, is the largest value of rank for
% homomorphisms, g, in argument list, Y, at any given time.
% Recursively traverse the list, Y, until it is empty, at
% which point MAXRANK binds with TEMPRANK.

for_each_g([],R,R).
for_each_g([homomorphism(R,K,Alg_Mn,Alg_M,Homo)|Rest],TR,MR) :-
    R >= TR, !,
    for_each_g(Rest,R,MR).
for_each_g([G|Gs],TR,MR) :-
    for_each_g(Gs,TR,MR).
% make_M_to_the_n(+ALG_M,+N,-ALG_MN)
% Given an algebra argument, ALG_M, and an integer argument, N,
% construct the algebra, ALG_MN, the Nth power of algebra, ALG_M.

make_M_to_the_n(algebra(M,FM),N,algebra(Mn,FMn)) :-
    make_universe(M,N,Mn),
    make_functions(FM,Mn,FMn).

% make_universe(+SET_M,+N,-SET_MN)
% Starts with set, SET_M, and counting up from 1 to N, builds each
% set, M to the power i, until reaching the set, SET_MN, which is
% SET_M to the power N.

make_universe(M,N,Mn) :- N =< 0, !, fail.
make_universe(M,1,M).
make_universe(M,N,Mn) :-
    N1 is N - 1,
    make_universe(M,N1,TempMn),
    make_universe_more(M,TempMn,TempMn,Mn).

% make_universe_more(+SET_M,+SET_MI,+SET_MI,-SET_MN); % Recursively traverses the set, SET_M, and adds each element of SET_M % to the end each element of the set, SET_MI, building the set, SET_MN. % One set argument, SET_MI, is used for traversing, while the other is % used as a starting point for each element in SET_M.

make_universe_more([],Set,Set,[]).
make_universe_more([X]|Rest],[],Set,Mn) :-
    make_universe_more(Rest,Set,Set,Mn).
    make_universe_more([[X]|Rest],Rest2,Set,Rest3).
I * FILE: subalgebra.pl
Does one of three things. 1 Checks that
ALG_B is a subalgebra of ALG_M, 2 generates
ALG_B a subalgebra of given ALG_M, or 3
checks or generates ALG_D where ALG_B' is
a subalgebra of ALG_D a subalgebra of ALG_M.

BY: Rich Little
CREATED: 4 Feb 1999
MODIFIED: 4 Mar 1999 (added subalgebra3)
7 Jan 2000 (added first clause of subalgebra3)

PREDICATES: subalgebra3
gen_subalgebra3
subalgebra2
closed2
generate_subalgebra4
apply_each_f/4
apply_f/4
make_function_list/3

CALLED-BY: rank/6, for_each_part/9, for_each_D/8,
for_each_i/4, is_algebra/1

% subalgebra(+ALG_B',-ALG_D,+ALG_M)
% If algebra argument, ALG_B', is the same as algebra argument,
% ALG_M, then ALG_D binds with ALG_B'. Else, The algebra, ALG_D, is
% generated such that ALG_D is a subalgebra of algebra, ALG_M, and
% algebra, ALG_B', is a subalgebra of ALG_D.

subalgebra(Alg,Alg,Alg), !.
subalgebra(Alg_Bprime,Alg_D,Alg_M) :-
   subalgebra(Alg_D,Alg_M),
   subalgebra(Alg_Bprime,Alg_D).

% gen_subalgebra(?X,-ALG_B,+ALG_M)
% Generate the smallest subalgebra, ALG_B, of algebra, ALG_M,
% that contains X. If X is a variable generate an arbitrary
% subalgebra.

gen_subalgebra(X0,algebra(B,FB),algebra(M,FM)) :-
   sublist(X0,M),
   generate_subalgebra(M,X0,FM,B),
   make_function_list(FM,B,FB).
% subalgebra(?ALG_B,+ALG_M)
% For a given algebra, ALG_B, checks if it is a subalgebra
% of algebra, ALG_M. Otherwise, generate an arbitrary subalgebra
% of algebra, ALG_M, which binds with ALG_B.

subalgebra(algebra(B,FB),algebra(M,FM)) :-
  sublist(B,M),
  closed(B,FM),
  make_functions_list(FM,B,FB).

% closed(+SET,+FM)
% Let FSET be the result of applying the functions of FM
% to the set, SET. SET is closed under FM if the union of SET
% and FSET equals SET. The empty set, [], does not generate
% a subalgebra.

closed([],FM) :- !, fail.
closed(XN,FM) :-
  list_to_ord_set(XN,OrdXN),
  apply_each_f(FM,OrdXN,[],FXN),
  list_to_ord_set(FXN,OrdFXN),
  ord_union(OrdXN,OrdFXN,XNplusl),
  ord_seteq(OrdXN,XNplusl).

% generate_subalgebra(+XN-1,+XN,+FM,-B)
% The empty set generates the empty subalgebra which
% is not considered an algebra. If the set, XN-1, is equal to
% the set, XN, we are done and B binds to XN. Else, let XN+1 be
% the union of XN and the resulting set from applying FM to XN
% and recurse on XN and XN+1.

generate_subalgebra([],A,B,C) :- !, fail.
generate_subalgebra(XNminus1,XN,FM,XN) :-
  ord_seteq(XNminus1,XN), !.
generate_subalgebra(XNminus1,XN,FM,B) :-
  list_to_ord_set(XN,OrdXN),
  apply_each_f(FM,OrdXN,[],FXN),
  list_to_ord_set(FXN,OrdFXN),
  ord_union(OrdXN,OrdFXN,XNplus1),
  generate_subalgebra(OrdXN,XNplus1,FM,B), !.

% apply_each_f(+FS,+X,+TEM_FX,-FINAL_FX)
% Apply each function in the list, FS, to the set, X, getting
% the set, FINAL_FX.

apply_each_f([],X,FsX,FsX).
apply_each_f([FM|Fs],X,FMX,FsX) :-
  apply_f(X,FM,FMX,TempFMX),
  apply_each_f(Fs,X,TempFMX,FsX).
% apply_f(+XS,+F,+FXS,-NEW_FXS)
%  Apply the function, F, to each element in the set, XS,
%  getting the set, NEW_FXS. Both lists, XS and F, are
%  ordered, thus only one pass through both is necessary.
apply_f([],FM,FMsX,FMsX).
apply_f([X|Xs],[f(X,FMX)|FM|FMs],FMsX,[FMX|FMsX]) :- 
  apply_f(Xs,FMs,FMsX,FMXs).
apply_f(Xs,[FM|FMs],FMsX,FMXs) :- 
  apply_f(Xs,FMs,FMsX,FMXs).

% make_functions_list(+FMS,+B,?FBS)
%  Construct the list of functions, FBS, where each
%  function in FBS is the restriction of the corresponding
%  function in the list, FMS, to the set B.
make_functions_list([],B,[]).
make_functions_list([FM|FMs],B,[FB|FBS]) :- 
  make_function_list(B,FM,FB), 
  make_functions_list(FMs,B,FBS), !.

% make_function_list(+BS,+FM,?FB)
%  For each element in list, BS, the result of applying
%  the function, FB, to the element is equal to the result
%  of applying the function, FM, to that element.
make_function_list([],FM,[]).
make_function_list([B|Bs],[f(B,FB)|FMs],[f(B,FB)|FBS]) :- 
  make_function_list(Bs,FMs,FBS), !.
make_function_list([B|Bs],[FM|FMs],FBS) :- 
  make_function_list([B|Bs],FMs,FBS), !.
FILE: homomorphism.pl
Tests for a homomorphism from an algebra B to an algebra M

BY: Rich Little
CREATED: 1 Nov 1999
MODIFIED: 2 Dec 1999 (added header and documentation)

PREDICATES: homomorphism l 4
extend_homomorphism l 5
gen_homol 5
build_hl 6
for_all_fl 6
build_on_Bl 5
preserves_operationsl 3
for_each_fl 4

CALLED-BY: lifts/3, is_homomorphism/1

% homomorphism(algebra(+B,+FB),algebra(+M,+FM),[+DOMPH,+PH],[?DOMH,?H])
% If the list, B, equals the list, DOMPH, and the homomorphism
% [DOMPH,PH] preserves the functions, FB and FM, then [DOMH,H] binds
% to the homomorphism, [DOMPH,PH]. Else, pick the next element
% in the list resulting from subtracting DOMPH from B, and extend
% the homomorphism.

% homomorphism(algebra(B,FB),algebra(M,FM),[H,HL],[H,HL]) :-
% ord_seteq(H,B),
% preserves_operations([H,HL],FB,FM).

% extend_homomorphism(+ELEMENTOF_B,+ALG_B,+ALG_M,+HOMO,-EXTENDED_HOMO)
% If extension is possible, then extend the homomorphism on element,
% ELEMENTOF_B and the first element of the universe of algebra, ALG_M.
% Else, recurse on ELEMENTOF_B and the next element of the universe
% of ALG_M, binding with EXTENDED_HOMO.

% extend_homomorphism(NB,algebra(B,FB),algebra([M|Ms],FM),[H,HL],[NewH,NewHL]) :-
% ord_add_element(H,NB,NH),
% ord_add_element(HL,h(NB,M),NHL),
% gen_homo([NH,NHL],[H,HL],algebra(B,FB),algebra([M|Ms],FM),[NewH,NewHL]).
% extend_homomorphism(NB,algebra(B,FB),algebra([M|Ms],FM),[H,HL],[NewH,NewHL]) :-
% extend_homomorphism(NB,algebra(B,FB),algebra(Ms,FM),[H,HL],[NewH,NewHL]).
% gen_homo(+HOMO_N+1,+HOMO_N,+ALG_B,+ALG_M,-HOMO)
% If the list, HOMO_N+1, equals list, HOMO_N, HOMO_N+1 binds
% with HOMO. Else, extend the homomorphism on all the elements
% in HOMO_N+1 not in HOMO_N. Let HOMO_N+2 be the union of HOMO_N+1
% and HOMO_N. We recurse on HOMO_N+2 and HOMO_N+1.

gen_homo([DHnpl1,Hnpl1],[DHn,Hn],algebra(B,FB),algebra(M,FM),[DHnpl1,Hnpl1]) :-
  ord_seteq(Hnpl1,Hn), !.

% build_H(+HOMO_N1\HOMO_N,+HOMO_N1,+FUNCTION_B,+FUNCTION_M,+PART_HOMO,-GEN_HOMO)
% Recurse on the list of elements, HOMO_N1\HOMO_N, in order to extend the
% partial homomorphism, PART_HOMO. When the list, HOMO_N1\HOMO_N, is empty,
% GEN_HOMO binds with PART_HOMO.

build_H([],Homo,FB,FM,Homol,Homol).
build_H([X|XS],Homo1,FB,FM,PartHomo,GenHomo) :-
  !, for_all_f(X,Homo1,FB,FM,PartHomo,TempHomo),
  !, build_H(XS,Homo1,FB,FM,TempHomo,GenHomo).

% for_all_f(+H(X,Y),+HOMO_N1,+FUNCTION_B,+FUNCTION_M,+PART_HOMO,-NEW_HOMO)
% Recurse through the function lists, FUNCTIONLIST_B and FUNCTIONLIST_M,
% applying each function of FUNCTIONLIST_B to the element, X, and applying
% each function of FUNCTIONLIST_M to the element, Y.

for_all_f(Elem,HomoN1,[],[],Homo,Homo).
for_all_f(h(X,Y),H1,[FB|FBs],[FM|FMs],Homo,NewHomo) :-
  find_f_of(X,FB,FofX),
  find_f_of(Y,FM,FofY),
  !, build_on_B(FofX,FofY,H1,Homo,THomo),
  !, for_all_f(h(X,Y),H1,FBs,FM,THomo,NewHomo).

% build_on_B(+NEWB,+NEWM,+[+DPH1,+PH1],[+DPH2,+PH2],[+DNH,+NH])
% If element, NEWB, is already in the list, DPH1, and h(NEWB,NEWM)
% is in the list, PH1, then [DNH,NH] binds to homomorphism, [DPH2,PH2].
% Else, if NEWB is in list, DPH2, and h(NEWB,NEWM) is in list, PH2,
% then [DNH,NH] binds to [DPH2,PH2]. Else, add NEWB to DPH2 and add
% h(NEWB,NEWM) to PH2 resulting in [DNH,NH].

build_on_B(NB, NM, [H,HL], [H1,HL1], [H1,HL1]) :-
  member(NB,H), !, member(h(NB,NM),HL).
build_on_B(NB, NM, [H,HL], [H1,HL1], [H1,HL1]) :-
  member(NB,H1), !, member(h(NB,NM),HL1).
build_on_B(NB, NM, [H,HL], [H2,HL1], [NewH1,NewHL1]) :-
  ord_add_element(HL1,h(NB,NM),NewHL1),
  ord_add_element(H1,NB,NewH1).
% preserves_operations(+HOMO,+FB,+FM)
% In order for HOMO to be a homomorphism, each function in the function
% lists, FB and FM, must be preserved for each element in the domain
% of HOMO.

preserves_operations([],Homo,FB,FM).
preserves_operations([X|Xs],Homo,FB,FM) :-
  for_each_f(FB,FM,X,Homo),
  preserves_operations([Xs,Homo],FB,FM).

% for_each_f(+FBS,+FMS,+X,+HOMO)
% The functions in the function lists, FBS and FMS, are considered to be
% preserved if HOMO(FB(X)) equals FM(HOMO(X)) for each function FM in FMS
% in FBS.

for_each_f([],[],X,Homo).
for_each_f([FB|F Bs],[FM|F Ms],X,Homo) :-
  find_f_of(X,FB,FBofX),
  find_h_of(FBofX,Homo,HofFBofX),
  find_h_of(X,Homo,HofX),
  find_f_of(HofX,FM,FMofHofX),
  HofFBofX == FMofHofX,
  !,
  for_each_f(FBs,F Ms,X,Homo).
**FILE: partition_k.pl**

Partitions fixed K into a list of size N, by first creating the list \([K,0,...,0]\), with \(N-1\) 0's, then giving it the tag \([1]\). The tag represents the rightmost nonzero position in the list. The next stage is to take the first partition and make new ones by creating the list \([K-i,i,0,...,0]\) for each \(i\) less than or equal to \(K\), until \(K=0\). Then we do the same for each of these in the 2 position, and so on until tag=N.

**BY:** Rich Little

**CREATED:** 30 April 1998

**MODIFIED:** 7 Jan 2000 (built-in append/3 replaced conc/3)

**PREDICATES:**

- partition_k/3
- first_partition/4
- rest_of_partitions/3
- build_on_partition/5
- new_partitions/2
- put_stripped_back/3
- add_tag/3
- lose_tag/2

**CALLED-BY:**

- for_each_k/9

% partition_k(+K,+N,-LISTOFPARTITIONEDK)
% Partition the integer, \(K\), into a list of size \(N\), binding with the argument, LISTOFPARTITIONEDK.

\[
\text{partition_k}(K,N,\text{NewKLists}) :- \\
\text{first_partition}(K,N,N,\text{KList}), \\
\text{rest_of_partitions}([\text{KList}],N,\text{KLists}), \\
\text{lose_tag}(\text{KLists},\text{NewKLists}). \\
\]

% first_partition(+K,+N,+COUNTER,-FIRSTPARTITION)
% Build a list, \([K,0,...,0]\) with \(N-1\) 0's, and add the tag \([1]\), getting a list of 2 lists, \([\[1\],[K,0,...,0]\]) which binds with argument, FIRSTPARTITION. The argument, COUNTER, is used to get the number of 0's needed.

\[
\text{first_partition}(K,N,0,[]). \\
\text{first_partition}(K,N,N,[[1],[K|Ks]]) :- \\
N1 \text{~is~} N-1, \\
\text{first_partition}(K,N,N1,Ks). \\
\]

132
first_partition(K,N,N1,[0|Ks]) :-
    N2 is N1-1,
first_partition(K,N,N2,Ks).

% rest_of_partitions(+PARTITIONS,+N,-EXTENDEDPARTITIONS)
% For each partition in list, PARTITIONS, build on it
% by pulling out the first element, adding it to the list,
% EXTENDEDPARTITIONS, and then passing it to build_on_partition/5.
% Add the new partitions onto the end of PARTITIONS and run
% through it recursively until it is an empty list.
rest_of_partitions([],N,[]).
rest_of_partitions([KList|R1],N,[KList|R2]) :-
    build_on_partition(KList,N,1,Stripped,TempKList),
    put_stripped_back(Stripped,TempKList,NewTempKList),
    append(R1,NewTempKList,NewKList),
    rest_of_partitions(NewKList,N,R2).

% build_on_partition((+TAG,+PARTITION),+N,+COUNTER,+STRIPPED,-BUILT)
% When tag, TAG, equals N, the partition PARTITION can no longer be
% built on. Otherwise, use COUNTER to move through PARTITION to get
% to the tag position, ie tag=COUNTER. Then make the new partitions
% and add their tags.
build_on_partition([[],List],N,1,[],[]).
build_on_partition([M,[F1,F2|R1]],N,M,[],NewKList) :-
    new_partitions([F1,F2|R1],KList),
    M1 is M+1,
    add_tag(M1,KList,NewKList).
build_on_partition([M,[F1,F2|R1]],N,M1,[F1|R2],R3) :-
    M2 is M1+1,
    build_on_partition([M,[F2|R1]],N,M2,R2,R3).

% new_partitions(+PARTITION,-NEWPARTITIONS)
% Subtract 1 from the first element of list, PARTITION,
% and add 1 to the second element of list, PARTITION,
% recursively until the first element is 0.
new_partitions([0|R1],[]).
new_partitions([F1,F2|R1],[[F3,F4|R1]|R2]) :-
    F3 is F1-1,
    F4 is F2+1,
    new_partitions([F3,F4|R1],R2).

% put_stripped_back(+STRIPPED,+EXTPARTITIONS,-PARTITIONS)
% When we extend our partitions, we move through
% the old partition to get to the tag position.
% As we do this, we strip off the front of the list.
% We put the stripped off partitions back on to the new
% list, EXTPARTITIONS, binding with PARTITIONS.
put_stripped_back([], List, List).
put_stripped_back(List, [], []).  
put_stripped_back(List, [[Tag, List1]|R1], [[Tag, List2]|R2]) :-
   append(List, List1, List2),
   put_stripped_back(List, R1, R2).

% add_tag(+TAG, +NEWPARTITIONS, -TAGGEDPARTITIONS)
% Recursively traverse the list, NEWPARTITIONS, inserting
% the element, TAG, at the head of each list.
add_tag([], []).  
add_tag(Tag, [List|R1], [[Tag, List]|R2]) :-
   add_tag(Tag, R1, R2).

% lose_tag(+TAGGEDPARTITIONS, -PARTITIONS)
% The argument, PARTITIONS, binds with the list,
% TAGGEDPARTITIONS, after the first element in each list
% of TAGGEDPARTITIONS is removed.
lose_tag([], []).  
lose_tag([[Tag, List]|R1], [List|R2]) :-
   lose_tag(R1, R2).
Using an algebra, B, and a fixed finite \( K \), create an algebra, \( B' \), by repetition of coordinates.

**BY:** Rich Little

**CREATED:** 7 April 1998

---

% embed_B(+ALGEBRA_B,+K,-ALGEBRA_B_PRIME)
% Embed algebra argument, ALGEBRA_B into an algebra by repetition
% of coordinates for finite integer argument, K. The new algebra
% binds with the uninstantiated argument, ALGEBRA_B_PRIME.

embed_B(algebra(B,FB) ,K,algebra(Bprime,FBprime)) :-
   rep_coords(B,K,Bprime),
   make_functions(FB,Bprime,FBprime).

% rep_coords(+SET_B,+K,-SET_B_PRIME)
% Walk the list, SET_B, of element lists, applying the same
% repetition of coordinates, K, to each. The new list binds
% with SET_B_PRIME.

rep_coords([],K,[]).
rep_coords([XjRest],K,[YjRest1]) :-
   rep_coords_each(X,K,Y),
   rep_coords(Rest,K,Rest1).

% rep_coords_each(+ELEMENT_OF_B,+K,-ELEMENT_OF_B_PRIME)
% The argument, K, is represented by a list of the same
% size as the element list, ELEMENT_OF_B. The elements of the
% list, K, are integers which. A 0 in the list means you do
% not repeat the corresponding coordinate in ELEMENT_OF_B,
% (clauses 3), a 1 means one copy of the coordinate, etc,
% (clause 2).

rep_coords_each([],[],[]).
rep_coords_each([X|Rest],[K|Rest1],[X,X|Rest2]) :-
   K > 0, !,
   K1 is K-1,
rep_coords_each([X|Rest],[K1|Rest1],[X|Rest2]).
rep_coords_each([X|Rest],[0|Rest1],[X|Rest2]) :-
    rep_coords_each(Rest,Rest1,Rest2).
/* FILE: lifts.pl */
/* Tests for lifting between an algebra, D, or factor algebra, C/Y to an algebra, M. */
/* BY: Rich Little */
/* CREATED: 16 Jan 1999 */
/* MODIFIED: */
/* */
/* PREDICATES: lifts/3 */
/* make_mew/3 */
/* find_X/3 */
/* */
/* CALLED-BY: for_each_D/8, there_exists_Y/7 */
/* */

% lifts(+HPRIME,+ALG,+ALG_M)
% A homomorphism, HPRIME, lifts to an algebra, ALG, if there exists
% a homomorphism, mew, from ALG to an algebra, ALG_M, extended from the
% natural homomorphism, i, from an algebra, B', to ALG such that
% mew(x) = h'(x), for all x in B'.

lifts([DomHprime,Hprime],Alg_D,Alg_M) :-
    make_mew(Hprime,Alg_D,PartHomo),

% make_mew(+HPRIME,+ALG,-PART_HOMO)
% Construct a potential partial homomorphism which binds with
% the uninstantiated argument, PART_HOMO.
make_mew([],Alg_D,[[],[]]).
make_mew([h(X,Y)|Rest],Alg_D,[[IofX|Rest1],[h(IofX,Y)|Rest2]]) :-
    find_X(X,Alg_D,IofX),
    make_mew(Rest,Alg_D,[Rest1,Rest2]).

% find_X(+X,+ALG,-IOFX)
% If element, X, is the next element of algebra, ALG, then IOF
% binds with X. Else, X is an element of the next class of factor
% algebra, ALG, then IOFX binds with that class. Else, recurse
% on the elements of ALG.
find_X(X,algebra([X|Xs],FL),X).
find_X(X,algebra([Tag,List]|Rest],FL),[Tag,List]) :-
    member(X,List).
find_X(X,algebra([Y|Ys],FL),IofX) :-
    find_X(X,algebra(Ys,FL),IofX).
% get_Ys(+ALG_M,+ALG_MN,+K,+N,-LISTOFYS)
% When using get_Ys/5, it is assumed that the database of algebraic
% operations of an algebra, ALG_M, with known rank are asserted as
% facts in the system. All instances of homomorphism/5 with
% arguments K, ALG_M, and ALG_MN are collected in an ordered set.
% The order of this set is based on the rank of the homomorphisms.
% Reverse the order of this set then collect all instances of
% sublists of the set in an ordered set. Each sublist is ordered
% from maximum rank to minimum. Finally, remove all the sublists
% that are strictly larger than N.

get_Ys(M,Mn,K,N,Ys) :-
  setof(homomorphism(R,K,Mn,M,H),homomorphism(R,K,Mn,M,H),Gs),
  reverse(Gs, RGs),
  setof(G, sublist(G, RGs), Zs),
  sizeof_g_less_than_N(Zs, N, Ys).

% get_Ys(+ALG_M,+ALG_MN,+K,+N,+FILE,-LISTOFYS)
% The predicate, get_Ys/6, does the same thing as get_Ys/5 except that
% the algebraic operations of ALG_M with known rank are stored in
% readable source file, FILE.

get_Ys(Alg_M,Alg_Mn,K,N,File,Ys) :-
  open(File, read, Stream),
  read(Stream, Term),
  find_gs(Term, Stream, K, Alg_M, Alg_Mn, Gs),
  close(Stream),
  list_to_ord_set(Gs, OGs),
  reverse(OGs, RGs),
  setof(G, sublist(G, RGs), Zs),
  sizeof_g_less_than_N(Zs, N, Ys).
% sizeof_g_lessthan_N(+LIST,+N,-WANTED_LIST)
%   Recursively remove all the elements of the list, LIST, that are
%   strictly larger than integer, N. The resulting list binds with
%   the argument, WANTED_LIST.

sizeof_g_lessthan_N([],N,[]).
sizeof_g_lessthan_N([Z|Zs],N,[Z|Ys]) :-
    count(Z,M),
    M =< N,
    M > 0,
    sizeof_g_lessthan_N(Zs,N,Ys).
sizeof_g_lessthan_N([Z|Zs],N,Ys) :-
    sizeof_g_lessthan_N(Zs,N,Ys).

% find_gs(+TERM,+STREAM,+K,+ALG_M,+ALG_MN,-GS)
%   Search the database attached to stream, STREAM, keeping only the
%   homomorphisms from algebra, ALG_MN, to algebra, ALG_M, for some, K.
%   The resulting list binds with argument, GS.

find_gs(end_of_file,S,K,Mn,[]).
find_gs(homomorphism(R,K,Mn,M,H),S,K,M,Mn,[homomorphism(R,K,Mn,M,H)|Rest]) :-
    read(S,Term),
    find_gs(Term,S,K,M,Mn,Rest).
find_gs(Term1,S,K,M,Mn,List) :-
    read(S,Term2),
    find_gs(Term2,S,K,M,Mn,List).
Checks if a set, \( Y \), of homomorphisms from an algebra, \( D \), to an algebra, \( M \), separates an algebra, \( B' \).

Rich Little  
2 Jan 1999  
19 Feb 1999 (removed get_Ys/4, in its own file)

PREDICATES: separates/3  
separated/1  
CALLED BY: there_exists_Y/7

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% separates(+Y,+ALGEBRA_BPRIME,-UNIVERSE_BPRIMEMODY)  
% Checks that the set of homomorphisms, \( Y \), separates an algebra, \( ALGEBRA_BPRIME \), by constructing the universe of the factor algebra \( ALGEBRA_B \) with respect to \( Y \). If the factor algebra is separated, then argument, \( UNIVERSE_BPRIMEMODY \), binds to the factor universe.

separates(Y,algebra(BPrm,FBPrm),Fact_Uni) :-  
  factor_universe(BPrm,Y,[],Fact_Uni),  
  !, separated(Fact_Uni).

% separated(+FACTORALGEBRA)  
% The factor algebra, \( FACTORALGEBRA \), is considered to be separated if each class in the algebra contains only one element. Done recursively on the classes of the factor algebra.

separated([]).  
separated([[Tag,Class]|Rest]) :-  
  count(Class,C),  
  !, C == 1,  
  separated(Rest).
Builds a factor algebra, \( C/Y \), given an algebra, \( C \), and a list of homomorphisms, \( Y \).

Rich Little
2 Jan 1999
22 Feb 1999 (changed \( \text{factor}_\text{universe} \) to \( 4 \))
(added \( \text{apply}_Y \) to \( 3 \))

PREDICATES: \( \text{factor}_\text{algebra} \), \( \text{factor}_\text{universe} \), \( \text{for}_\text{each}_\text{function} \), \( \text{factor}_\text{function} \), \( \text{apply}_Y \), \( \text{add}_\text{element} \), \( \text{in}_\text{class} \)

CALLED-BY: \( \text{there}_\text{exists}_Y \), \( \text{factor}_\text{function} \), \( \text{add}_\text{element} \), \( \text{in}_\text{class} \)

% \( \text{factor}_\text{algebra}( +\text{ALG}_C, +Y, +\text{ALG}_BPRIME, +BPRIME, Y, \text{-ALG}_C/Y) \)
% Creates the factor algebra of an algebra, \( \text{ALG}_C \), with respect to the homomorphism list, \( Y \), and binds it to the argument, \( \text{ALG}_C/Y \).
% The argument, \( \text{ALG}_BPRIME \), represents a subalgebra of \( \text{ALG}_C \) which has a universe that has been previously factored as, \( BPRIME/Y \).

\[ \text{factor}_\text{algebra}(\text{algebra}(C,FC), Y,\text{algebra}(Bprm,FBprm),BprmY,\text{algebra}(CY,FCY)) :- \]
\[ \text{ord}_\text{subtract}(C,Bprm,\text{RestofC}), \]
\[ \text{factor}_\text{universe}(\text{RestofC},Y,Bprm,CY), \]
\[ \text{for}_\text{each}_\text{function}(CY,FC,FCY). \]

% \( \text{factor}_\text{universe}( +\text{UNI}_C \backslash \text{UNI}_BPRIME, +Y, +\text{UNI}_BPRIME, Y, \text{-UNI}_C/Y) \)
% Recurse through the list of elements, \( \text{UNI}_C \backslash \text{UNI}_BPRIME \), applying the set of homomorphisms, \( Y \), to each element in order to add each element to the appropriate class. The argument, \( \text{UNI}_C/Y \), binds to the list, \( \text{UNI}_BPRIME/Y \), once all the elements are the appropriate class.

\[ \text{factor}_\text{universe}([\text{}], Y,\text{Universe},\text{Universe}). \]
\[ \text{factor}_\text{universe}([X|Xs], Y,\text{Universe},\text{NewUniverse}) :- \]
\[ \text{apply}_Y(Y,X,Tag), \]
\[ \text{add}_\text{element}(X,Tag,\text{Universe},\text{TempUniverse}), \]
\[ \text{factor}_\text{universe}(Xs,Y,\text{TempUniverse},\text{NewUniverse}). \]

% \( \text{for}_\text{each}_\text{function}( +\text{UNI}_C/Y, +\text{FUNCTIONLIST}_C, \text{-FUNCTIONLIST}_C/Y) \)
% For each function in the function list, \( \text{FUNCTIONLIST}_C \), create the corresponding function for the factor algebra with underlying universe, \( \text{UNI}_C/Y \), binding with \( \text{FUNCTIONLIST}_C/Y \).
for_each_function(CmodY,[],[]).
for_each_function(CmodY,[F|Fs],[FCmodY|FCmodYs]) :-
  factor_function(CmodY,CmodY,F,FCmodY),
  for_each_function(CmodY,Fs,FCmodYs).

% factor_function(+UNI_C/Y,+UNI_C/Y,+FUNCTION_C,-FUNCTION_C/Y)
% Recurse on list, UNI_C/Y, the universe of a factor algebra, in
% order to create the function list which corresponds to this factor
% universe, which binds with the argument, FUNCTION_C/Y.
factor_function([],UniCY,FC,[]).
factor_function([[[Tag,X|[X|Xs]],|R]],UniCY,FC,[f([[Tag,[X|Xs]],|Class]]|R1)) :-
  find_f_of(X,FC,FofX),
in_class(FofX,UniCY,Class),
factor_function(R,UniCY,FC,R1).

% apply_Y(+Y,+X,-TAG)
% Recursively apply each homomorphism in a list, Y, to an element, X,
% resulting in a list which binds with argument TAG, to be used
% as a key for the classes of a factor algebra.
apply_Y([],X,[]).
  find_h_of(X,Homo,GofX),
  apply_Y(Gs,X,Rest).

% add_element(+ELEMENT,+TAG,+UNIVERSE,-NEWUNIVERSE)
% If class with key name, TAG, does not exist, then create it,
% add the element, ELEMENT, to that class and add that class to
% the list, UNIVERSE, binding it with NEWUNIVERSE. Else, add ELEMENT
% to the appropriate class by recursing on the elements (classes) of
% argument, UNIVERSE, binding the resulting list with NEWUNIVERSE.
add_element(Element,Tag,[],[[Tag,[Element]]]).
add_element(Element,Tag,[[Tag,Class]|Rest],[[Tag,[Element|Class]]|Rest]).
add_element(Element,Tag,[[Class|Classes],|Rest]) :-
  add_element(Element,Tag,Classes,Classes1).

% in_class(+ELEMENT,+UNI_C/Y,-CLASS)
% If the element, ELEMENT, does not belong to list in the list,
% UNI_C/Y, then in_class/3 fails. Else, recurse on UNI_C/Y, until
% the class list containing ELEMENT is found and is bound to argument,
% CLASS.
in_class(Element,[],Class) :- !, fail.
in_class(Element,[[Tag,Class]|Rest],[Tag,Class]) :-
  member(Element,Class), !.
in_class(Element,[[Class|Classes],|EqClass]) :-
  in_class(Element,Classes,EqClass).

142
% make_projections(+ALG_M,+I,+K,+FILE)
% Create a database of projections from subalgebras of finite
% powers of algebra, ALG_M, to ALG_M, for 1 \leq I \leq K. Store the
% database as a file with filename, FILE.

make_projections(algebra(M,FM),I,K,File) :-
    open(File1,write,Stream),
    list_to_ord_set(M,OM),
    ord_functions(FM,OFM),
    for_each_i(algebra(OM,OFM),K,I,Stream),
    close(Stream).

% for_each_i(+ALG_M,+K,+I,+STREAM)
% For each integer, I, from 1 to some integer, K, construct the Ith
% power of algebra, ALG_M, and collect all the subalgebras
% of the resulting algebra.

for_each_i(Algebra_M,K,I,Stream) :-
    I=<K, !,
    make_M_to_the_n(Algebra_M,I,Algebra_Mi),
    bagof(Algebra_B,subalgebra(Algebra_B,Algebra_Mi),Subalgebras),
    for_each_sub(Subalgebras,Algebra_M,I,Stream),
    I1 is I+1,
    for_each_i(Algebra_M,K,I1,Stream).

% for_each_sub(+SUBS,+ALG_M,+I,+STREAM)
% Recursively traverse the list of algebras, SUBS.
for_each_sub([], Algebra_M, I, Stream).
for_each_sub([Alg_B | Algs_B], Algebra_M, I, Stream) :-
  for_each_j(Algebra_M, I, Alg_B, 1, Stream),
  for_each_sub(Algs_B, Algebra_M, I, Stream).

% for_each_j(+ALG_M, +I, +ALG_MI, +J, +STREAM)
% For each integer, J, from 1 to integer, I, construct the list
% of projections from an algebra, ALG_MI, to algebra, ALG_M,
% write them onto the stream, STREAM, as clauses of the
% predicate, homomorphism/5.
for_each_j(Algebra_M, I, Algebra_Mi, J, Stream) :-
  J=<I, !.
  projections(Algebra_Mi, J, Homomorphism),
  write(Stream, homomorphism(0, 'K', Algebra_Mi, Algebra_M, Homomorphism)),
  format(Stream, "-w", [', ']),
  format(Stream, "-n", []),
  format(Stream, "-n", []),
  J1 is J+1,
  for_each_j(Algebra_M, I, Algebra_Mi, J1, Stream).
for_each_j(Algebra_M, I, Algebra_Mi, J, Stream).

% projections(+ALG, +J, -PROJ)
% Argument, PROJ, binds with the homomorphism list created by
% taking the Jth projection of each element in an algebra, ALG.
projections(algebra([], Functionlist), J, [[], []]).
projections(algebra([X|Xs], Functionlist), J, [[X|Rest1], [h(X,Y)|Rest]]) :-
  proj(J, X, Y),
  projections(algebra(Xs, Functionlist), J, [Rest1, Rest]).

% proj(+N, +ELEMENT, -P)
% The argument, ELEMENT, is instantiated with an element list. If
% the integer, N, is strictly larger than the size of the list,
% ELEMENT, then proj/3 fails. Otherwise, argument, P, binds with
% the Nth projection of an element, ELEMENT.
proj(N, [], P) :- !, fail.
proj(1, [X|Rest], [X]).
proj(N, [X|Rest], P) :-
  N1 is N-1,
  proj(N1, Rest, P).
FILE: tools.pl

This file contains the shared tools needed by the predicates of other files

BY: Rich Little
CREATED: 10 Nov 1999
MODIFIED: 7 Jan 2000 (argument removed from make_hprime/3)
7 Jan 2000 (call to find_h_of/3 removed from same)

PREDICATES: make_functions/3
make_function/3
make_function_more/4
find_f_of/3
find_h_of/3
make_hprime/3
count/2
is_algebra/1
is_homomorphism/1
ord_functions/2
check_dBase/7

% make_functions(+FUNCTIONLIST_M,+SET_MN,-FUNCTIONLIST_MN);
% Recursively traverse the list of functions, FUNCTIONLIST_M,
% to generate a list of functions for the set, SET_MN, which
% binds with the argument, FUNCTIONLIST_MN.

make_functions([],Mn,[]).
make_functions([[FM|FMns],Mn,[FMn|FMns]] : -
make_function(Mn,FM,FMn),
make_functions(FMs,Mn,FMns)).

% make_function(+SET_MN,+FUNCTION_M,-FUNCTION_MN)
% Recurse on the elements of the list, SET_MN, generating a
% function list corresponding to the function list, FUNCTION_M,
% which binds with the argument, FUNCTION_MN.

make_function([],FM,[]).
make_function([X|Rest].FM,[Y|Rest1]) : -
make_function_more(X,FM,FM,Y),
make_function(Rest,FM,Rest1).

% make_function_more(+X,+FUNCTIONLIST_M,+FUNCTIONLIST_M,-Y)
% The argument, X, is an element list which is traversed recursively
% in order to generate the list of elements that X is mapped to by
% the function, FUNCTION_M. This new list binds with argument, Y.
% find_f_of(+X,+FUNCTIONLIST,-Y)
% Recurse on the elements of list, FUNCTIONLIST, looking for an
% instance of the predicate, f/2, where the first argument of f/2
% binds with the argument, X. If no such instance of f/2 exists, then
% find_f_of/3 fails. Otherwise, the argument, Y, binds with the second
% argument of f/2.
find_f_of(X,[],Y) :- !, fail.
find_f_of(X,[f(X,Y)|Rest],Y) :-
    find_f_of(X,Rest,Y).

% find_h_of(+X,+HOMOMORPHISM,-Y)
% Recurse on the elements of list, HOMOMORPHISM, looking for an
% instance of the predicate, h/2, where the first argument of h/2
% binds with the argument, X. If no such instance of h/2 exists, then
% find_h_of/3 fails. Otherwise, the argument, Y, binds with the second
% argument of h/2.
find_h_of(X,[],Y) :- !, fail.
find_h_of(X,[h(X,Y)|Rest],Y) :-
    find_h_of(X,Rest,Y).

% make_hprime(+HOMO_B,+ALG_BPRIME,-HOMO_BPRIME)
% Construct a new homomorphism from a homomorphism, HOMO_B,
% and an algebra, ALG_BPRIME. The argument, HOMO_BPRIME, binds
% to the new homomorphism.
make_hprime([DH,[]],algebra([],FBprm),[],[]).
make_hprime([DH,[h(X,Z)|Hs]],algebra([Y|Ys],FBprm),[[Y|Ys],[h(Y,Z)|HPs]]) :-
    make_hprime([DH,Hs],algebra(Ys,FBprm),[Ys,HPs]).

% count(+LIST,-SIZE)
% The argument, SIZE, binds with the number of elements in
% a list, LIST.
count([],0).
count([X|Xs],N) :-
    count(Xs,N1),
    N is N1+1.
% is_algebra(+ALGEBRA)
% The argument, ALGEBRA, is an algebra if it is a subalgebra of itself.

is_algebra(Algebra) :-
  subalgebra(Algebra, Algebra).

% is_homomorphism(+HOMOMORPHISM)
% Checks that argument, HOMOMORPHISM, represents a homomorphism.

is_homomorphism(homomorphism(Rank, K, Alg_B, Alg_M, Homo)) :-

% ord_functions(+FS,-ORDFS)
% For a list of lists, FS, generate a list of ordered sets, binding it with argument, ORDFS.

ord_functions([], []). ord_functions([F | Fs], [OF | OFs]) :-
  list_to_ord_set(F, OF),
  ord_functions(Fs, OFs).

% check_dBase(+TERM,+STREAM,+K,+HOMO,+ALG_B,+ALG_M,-RANK)
% If a homomorphism, HOMO, appears in a database given by STREAM, then the rank of the homomorphism binds with RANK.
% Else, the rank is unknown.

check_dBase(Term1, S, K, H, Alg_B, Alg_M, Rank) :-
  read(S, Term2),
  check_dBase(Term2, S, K, H, Alg_B, Alg_M, Rank).