HARVARD UNIVERSITY

DOCTORAL THESIS

A Language of Polynomials

Author:
Eric UNG

Supervisor:
Dr. xxx XXX

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Research Group Name
Department or School Name

June 27, 2024

Declaration of Authorship

I, Eric UNG, declare that this thesis titled, "A Language of Polynomials" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

HARVARD UNIVERSITY

Abstract

Faculty Name
Department or School Name

Doctor of Philosophy

A Language of Polynomials

by Eric UNG

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

For/Dedicated to/To my…

Chapter 1

A Language of Polynomials

1.1 Introduction

This paper is on the construction of the one way function to a matrix multiplication problem - that of multiplying two 3×3 matrices to form a 6×6 matrix under a locally concatenative property. The matrix multiplication is a type of law of composition that operates on different layers. There may be some higher level mathematics mixed throughout the paper, however, thorough explanation will be attempted, otherwise can be ignored. The reason why it is included is to provide a bridge of multiple directions for the reader to take.

This paper aims to get the reader up to speed on the foundations of current computational complexity theory as it builds up a framework from scratch. At the end it provides a field for the reader to use in order to understand modern complexity theory in more depth. The intention is to provide as much insight as possible as to explore the science of computation.

1.2 Foundations

There exists a language such that it decides each monomial in the polynomial. In other words, there exists a set of deciders for each monomial in the polynomial where it decides if y is in the monomial. A decider in this term is not of the definition found originally in textbooks but one that is redefined in the below definition.

Given a polynomial
$p(x) = ax^2 + bx + c$
$p(x) = 3x^2 + 4x + 5$
$p(2) = 3(2)^2 + 4(2) + 5$
$p(2) = 12 + 8 + 5$

Let the decider be defined as the following:

Decider is a function $Decider < cx^n > \equiv cx^n = y$
such that $x_1 \times x_2 \times ... \times x_n$ where n is equal to degree + constant is tested to be equivalent to y and x_1 is the start and x *degree times* is the finish then loop around $x_1 tox$ *n times* until it stops

For each state, x_i, i such that it is between 1 to n, x_i contains a subgroup of size n and for each subgroup, s_i, there exists another subgroup and so on and so forth

such that there are n layers starting from x_i to 1. This is the same as saying that it is a rational expression.

Examples
Decider for ax^2 is $Decider < 3(2)^2 >\equiv 3(2)^2 = 12$
Decider for bx is $Decider < 4(2)^1 >\equiv 4(2)^1 = 8$
Decider for c is $Decider < 5(2)^1 >\equiv 5(2)^0 = 5$

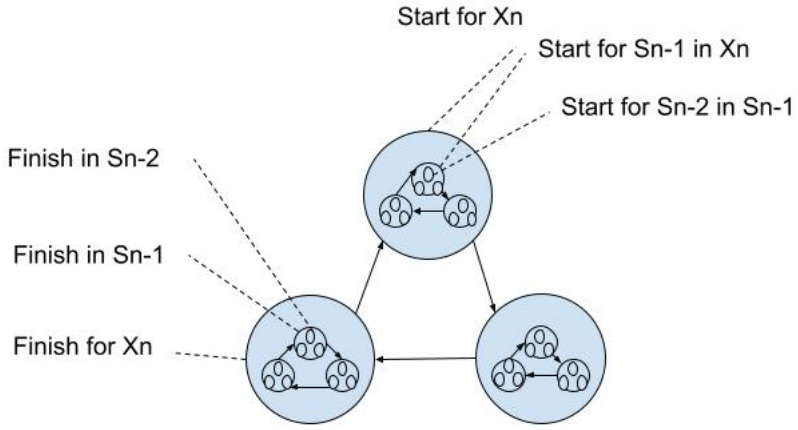

FIGURE 1.1: Decider that represents the monomial, x^3.

Theorem: A Decider is the equivalent to a cyclic automata

Proof: Remove the lowest level state, S_1 from the bottom then continue removing S_i from i = 2 to n-1 until you get only the states that are at X_n.

Remove the top state down from the S_{n-1} for each layer S_{n-1} to $S_(n-n-1)$. This preserves the start and finish state for the layer x_n. This is a cyclic automaton.

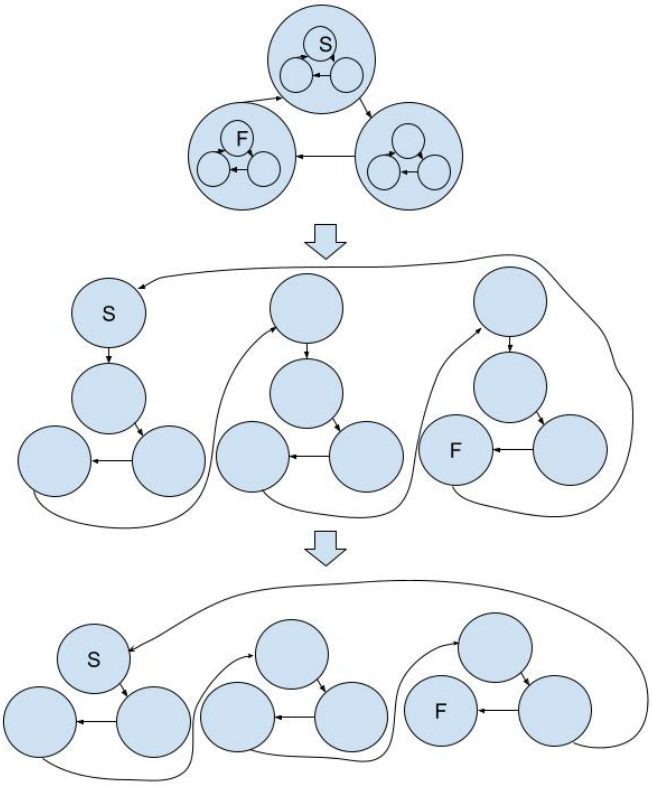

FIGURE 1.2: Top down removal for equivalence of decider and cyclic automata.

1.3 Monomial of One Variable

Given the definition of a decider:
Decider is a function $Decider < cx^n > \equiv cx^n = y$

A decider of at least one degree
$Decider < 3x^4 > \equiv 3x^4 = y$

Contains $Decider < 3x^3 > \equiv 3x^3 = y$
Contains $Decider < 3x^2 > \equiv 3x^2 = y$
Contains $Decider < 3x^1 > \equiv 3x^1 = y$
Contains $Decider < 3x^0 > \equiv 3x^0 = y$
Hence it can be generalized to:
$Decider < cx^n >$ contains the sequence set

There exists a start state and a finish state for each decider.
$\{start, ..., finish\}$

$Decider < cx^n >, Decider < cx^{n-1} >, ..., Decider < cx^0 >$ which has a more formal definition called a rational expression. A rational expression on A over K is a

semiring described as $\mathcal{E}_n$ such that $n \geq 0$ where A is an alphabet (in our case a finite set of integers) and K is a commutative semiring. This means the following in terms of the decider

$Decider < cx^n > = \mathcal{E}_n$
Contains $Decider < cx^{n-1} > = \mathcal{E}_{n-1}$
...
Contains $Decider < cx^1 > = \mathcal{E}_1$
Contains $Decider < cx^0 > = \mathcal{E}_0$

The formal definition of a rational expression is defined below.

Definition. $A_n = A_{n-1} \cup \{E^* | E \in \mathcal{E}_{n-1}, (E, 1) = 0\}$

Here, A_n is the set of monomials in the polynomial. A_{n-1} are the monomials of degree n-1 and less of the polynomial and the set $\{E^* | E \in \mathcal{E}_{n-1}, (E, 1) = 0\}$ is equivalent to $Decider < cx^n >$ or equivalently the top of a single state of a decider. This theory has the opinion that it tries to describe finite recursion and nested for loops in a visual manner and, in so doing, this chain is chosen to be the best definition.

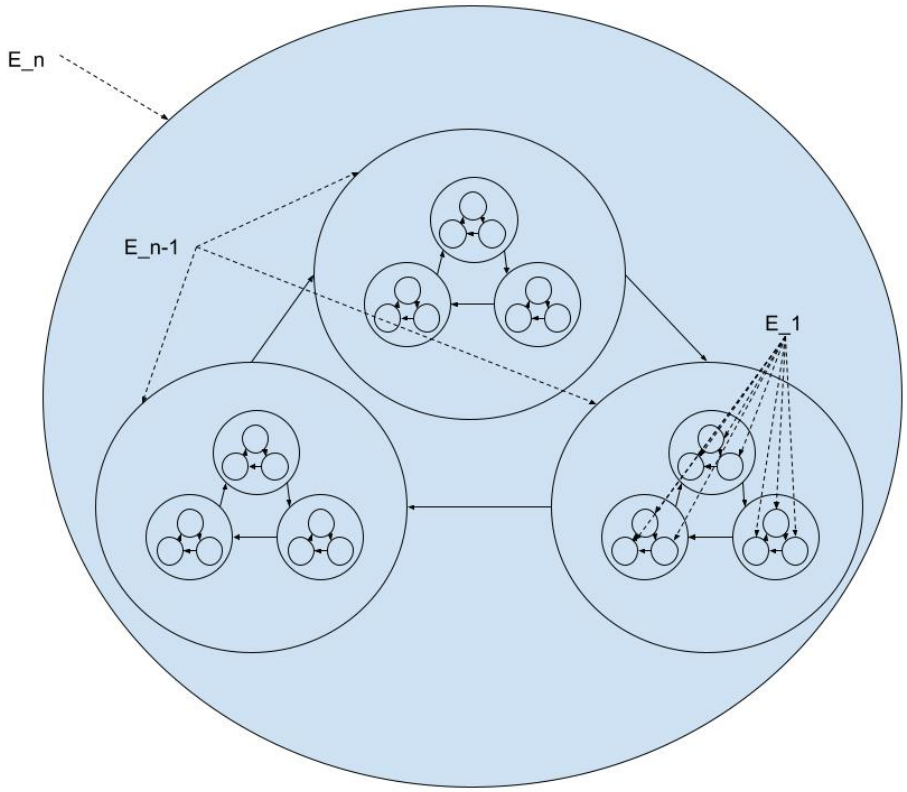

FIGURE 1.3: Visual example of what $\mathcal{E}_n$ of a rational expression.

Figure 1.3 Visual example of $\mathcal{E}_n$ of a rational expression.

A rational function is defined as the following:

K[x] and K[[x]]. Let K[[x]] describe a set of deciders as a polynomial representation. S is an element of K[[x]] meaning S is a decider.

$S = \sum_{n \geq 0} a_n x^n$

1.4 Addition

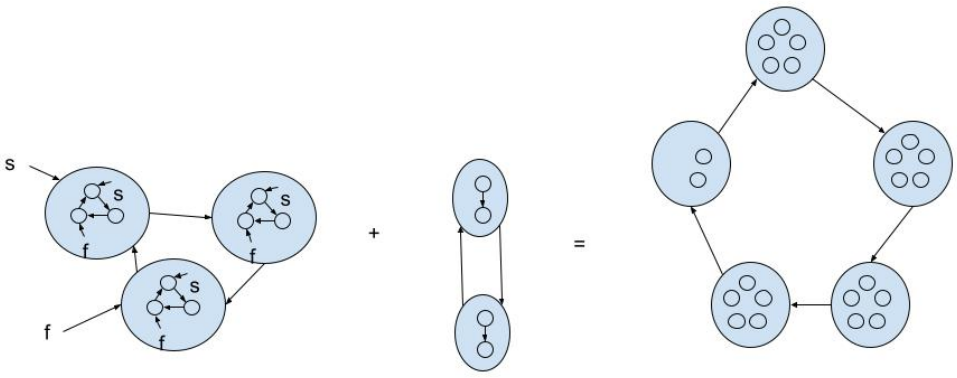

FIGURE 1.4: Addition of two deciders. The gradient of the circles remain the same after adding the two deciders together as the degree remains the same.

Given the first example:

$$p(x) = 3x^2 + 4x + 5$$
$$p(2) = 3(2)^2 + 4(2) + 5$$
$$p(2) = 12 + 8 + 5$$

$m2 = Decider < 3x^2 >= 3x^2$
$m1 = Decider < 4x^1 >= 4x$
$m0 = Decider < 5x^0 >= 5$

Generalized to m_x where x is the degree
Given polynomial functions, p_1 and p_2, they are commutative
$p_1(x) = m_a + ... + m_0$
$p_2(x) = n_b + ... + n_0$
$p_1(x) + p_2(x) = m_a + n_b + (m_{x+1} + n_{y+1}) + ... + (m_x + n_y) + ... + (m_0 + n_0)$ where $x = y$
$Decider < c_x x d_x > + Decider < c_y x d_y >$
$= Decider < c_x + c_y, x, d_x >= Deciders < c_x + c_y, x, d_y >$
$\implies d_x = d_y$

1.5 Product

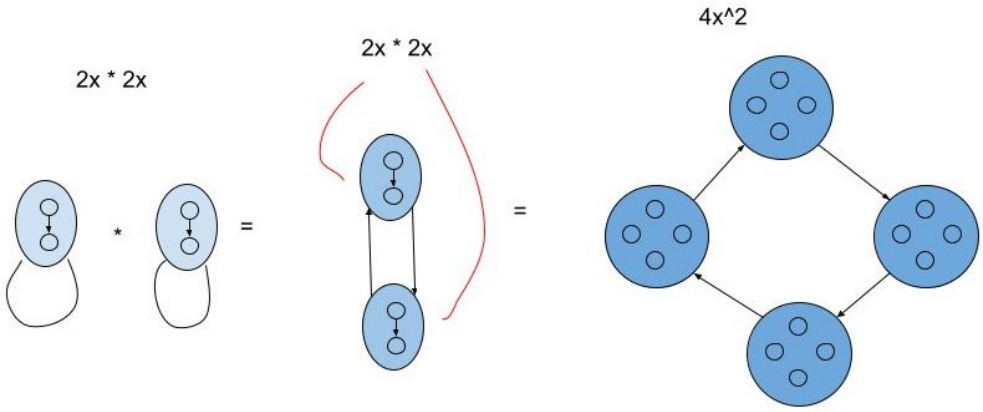

FIGURE 1.5: Product of two deciders. The gradient of the circles get denser after adding the two deciders together as the degree increases.

Given two monomials in the language, a and b, the product of a and b is also in the language.

Given $Decider < c_x x d_x >$ and $Decider < c_y x d_y >$ is in language L
Show that the product $Decider < (c_x + c_y) x d_x \times d_y >$ is in L
$Decider < c_x x d_x > \times Decider < c_y x d_y >$
$= c_x x d_x c_y x d_y$
$= c_x x d_x + d_y$
$= (c_x + c_y) x (d_x + d_y)$ is in L
$= Decider < (c_x + c_y) x (d_x + d_y) >$

1.6 Problem with Matrices

An important problem arising from deciders is representing them as matrices. The problem can be reformulated as the following: given a polynomial p of x, show that the monomial deciders represented in the language can't be contained in a finite matrix after a set number, n, such that x^n.

$[n \times n]\,[n \times n] = [m \times m]$ such that $m \neq n$ and $m, n \geq 0$ and $m \geq n$

The focus of this article pertains to the question of whether or not there exists structures with certain properties that allow law of compositions to handle the above statement. The reason why this seems feasible is because of the following proposition found in Retenaur(66).

Proposition. Given a proper square matrix M over $\mathcal{E}$, there exist matrices M_1, M_2 of the same size as M over $\mathcal{E}$ such that $M_1\ 1 + MM_1$ and $M_2\ 1 + M_2M$. In particular if K is a ring, 1 - M is an invertible modulo .

1.7 Multivariable Monomials

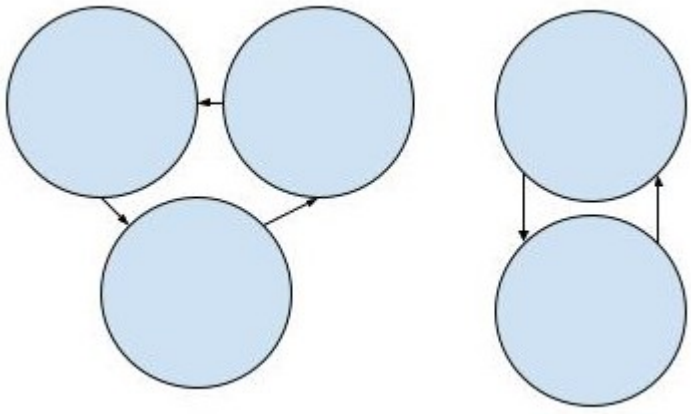

FIGURE 1.6: Multivariable monomial deciders can be seen treated as parallel processes running next to each other.

A monomial with more than one variable can be treated the same way as handling single variables at different degrees.

Addition gives the following:

$Decider < x^6yz^3 > + Decider < x^6yz^3 >= Decider < 2x^6yz^3 >$

Multiplication of the decider of the same degree gives the following:

$Decider < cx^n > \times Decider < cy^n > \times Decider < cz^z >$
$\equiv Decider < cx^n * cy^n * cz^n >$
$\equiv Decider < cxyz^{3n} >>$
where c is some constant

Multiplication of the decider of the different degrees gives the following:

$Decider < cx^n > \times Decider < cy^m > \times Decider < cz^l >$
$\equiv Decider < cx^n * cy^m * cz^l >$
$\equiv Decider < cx^{n+m+l} >$
where c is some constant

Given $Decider < 3xy^2 >$ and $Decider < 7x^7y^{-1} >$

$Decider < 3xy^2 > \times Decider < 7x^7y^{-1} >= Decider < 21x^8y >$

Representing a multivariable monomial of different degrees is a similar line of thought. There are many representations of them, however, this article will choose the simplest and have them separate as seen in figure 1.6. In this manner, multivariable monomials are a set of individual one variable monomials running simultaneously.

1.8 Generalized Monomial Deciders

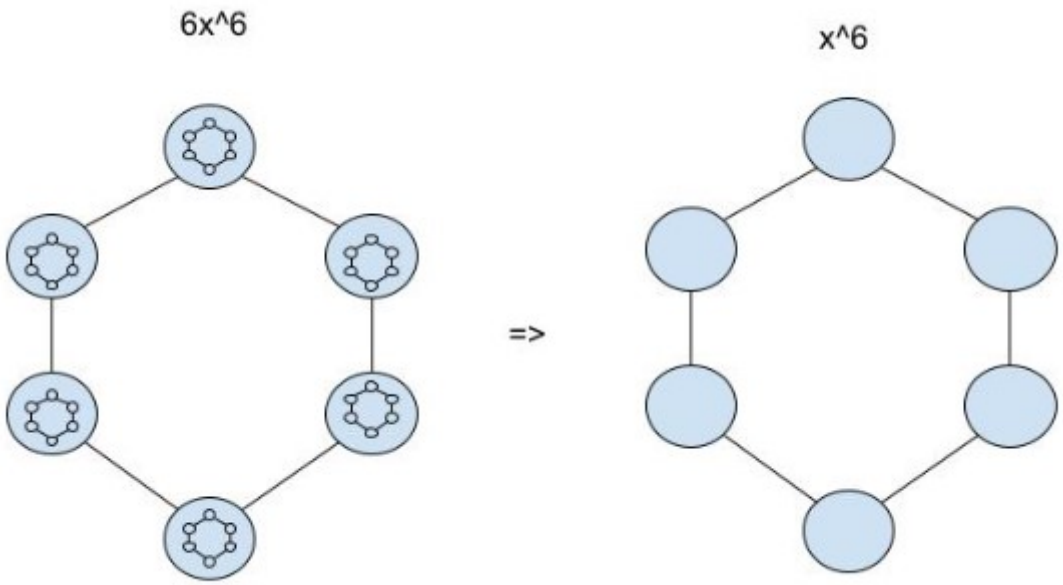

FIGURE 1.7: Generalization of a monomial decider.

A decider can be represented as the top layer only if short hand notation is necessary. In a way, this removes the constant which is a term called being *proper*, it isn't necessary true all the time.

Given a Decider<m(x)> where m(x) is a monomial, keep the top layer S_n in $\mathcal{E}_{\setminus}$. This is called the generalized monomial decider.

1.9 Concentric Monomial Deciders

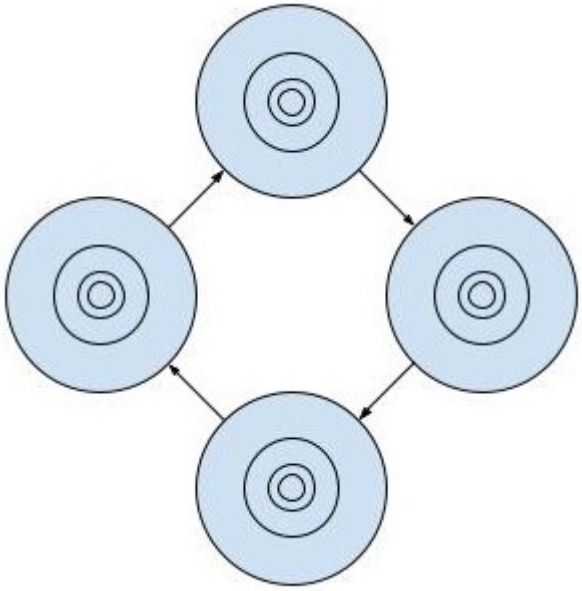

FIGURE 1.8: A concentric monomial decider is a generalized monomial decider with details missing.

Generalization results in an interesting property if monomial decider is required to get in more depth. The top layer that remains from generalization remains the same and still forms a cycle, however, each state has one state and so forth up to n-1 depth.

1.10 Constants

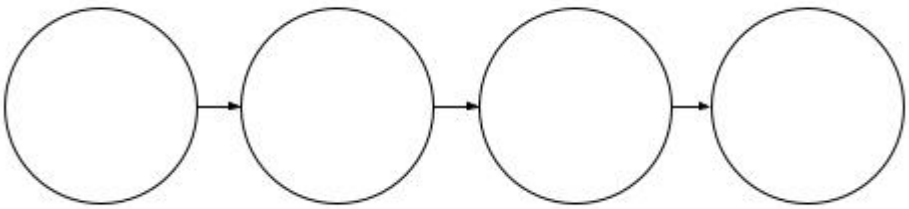

FIGURE 1.9: A constant represented as monomial decider, $Decider < cx^0 >$.

Given a constant, c, of a polynomial: f(x) = c, Constants are seen as linear directed acyclic graphs.

$$Decider < cx^0 > \equiv c = y$$

Addition gives the following:
$Decider < c_1x^0 > + Decider < c_2x^0 > \equiv Decider < (c_1 + c_2)x^0 > \equiv Decider < c_1 + c_2 >$

Multiplication gives the following:
$Decider < c_1x^0 > \times Decider < c_2x^0 > \equiv Decider < c_1c_2x^0 > \equiv Decider < c_1c_2 >$

There is no state in the decider where it loops back to the start. The constant is what separates a decider from a strictly mathematical cyclic object, semi-simple groups. This paper will mainly cover the cyclic part of the language.

1.11 Division

There are a finite amount of permutations, called decision functions, in a decider.

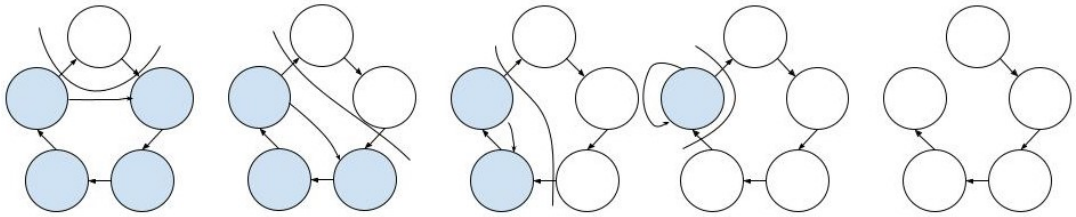

FIGURE 1.10: One decision function of $Decider < x^5/x >$ to one decision function of $Decider < x^5/x^5 >$.

Example. The following are deciders related to x^5/x^i such that $0 \leq i \leq 5$.
$Decider < x^5/x > \equiv Decider < x^5 > / Decider < x^1 >$
$Decider < x^5/x^2 > \equiv Decider < x^5 > / Decider < x^2 >$
$Decider < x^5/x^3 > \equiv Decider < x^5 > / Decider < x^3 >$
$Decider < x^5/x^4 > \equiv Decider < x^5 > / Decider < x^4 >$
$Decider < x^5/x^5 > \equiv Decider < x^5 > / Decider < x^5 >$

1.12 Multiple Divisions

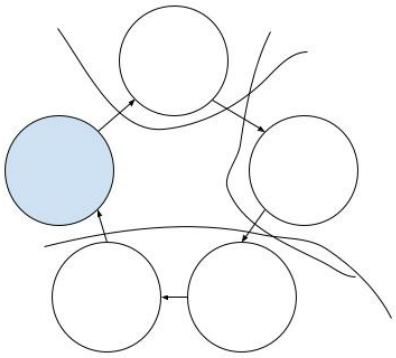

FIGURE 1.11: *Decider* $< x^5/x/x/x^2 >$

Given multiple operations of division, this forms a topological space where the order of operations are ignored.

Decider $< x^5/x^2/x/x >$ *Decider* $< x^5/x^2/x^2 >$.
Decider $< x^5/x^2/x^1/x^1 >$ *Decider* $< x^5/x^2/x^2 >$

The topology of the operations on the cyclic structure of a decider forms its own area of study. A topology of a decider used in this paper is constructed below.

Definition. A topology on a set X is a collection τ of subsets of X having the following properties:
1. $\emptyset$ and X are in τ
2. The union of elements of any sub-collection of τ is in τ
3. The intersection of the elements of any finite sub-collection of τ is in τ

A set X for which a topology τ has been specified is called a **topological space**. These three properties are used as template to construct a decider.

A decider of a monomial can have no decision functions or a set of decision functions so $\emptyset \in \tau$ or $D \subseteq \tau$

The union of sets of decision functions, D, of τ is in τ.

$d_1, d_2 \in D \subseteq \tau$ such that $d_1 \cup d_2 \subseteq D \subseteq \tau$ implies $d_1, d_2 \in \tau$.

The intersection of sets of decision functions, D, of τ is in τ.
Given $d_1 \in D_1$ and $d_1 \in D_2$, then $d_1 \in D_1 \cap D_2 \subseteq \tau$.

Using the construction above, a general categorization of the states of a decider can be illustrated to find basic patterns using a term in topology called the basis.

Definition. If X is a set, a basis for topology on X is a collection B of subsets of X such that:
1. For each $x \in X$, there is at least one basis element B containing x
2. If x belongs to the intersection of two basis elements B_1 and B_2, then there is a basis element B_3 containing x such that $B_3 \subset B_1 \cap B_2$.

A decision function has a start state and two possible outcomes, accept and reject. It can be generalized that there are three kinds of states - start, accept, and reject. These form a basis - B_{start}, B_{accept}, and B_{reject}. Each basis element has at least one state. A state can be in the basis elements of start and accept or start and reject meaning that the intersection forms another basis element - the intersection of two basis elements. Thus, this is the topology of a decision function generated by B_{start}, B_{accept}, and B_{reject}.

The next basis that can be formed is one that describes the operations on $x^{n+k}/Q(x^k)$ where Q is the divisions (e.g. $x^6/x^3/x^2$ where $Q(x^k) = x^3/x^2$). There are $6 * 2 + 6 * 1 = 18$ permutations possible in the example given. The basis elements that form B_{6*2} and B_{6*1} can be described. As a programmer, this statement means that even though there are many ways to write a function, there a finite amount of different combinations to write them in terms of having be as re-factored well.

1.13 Equivalence

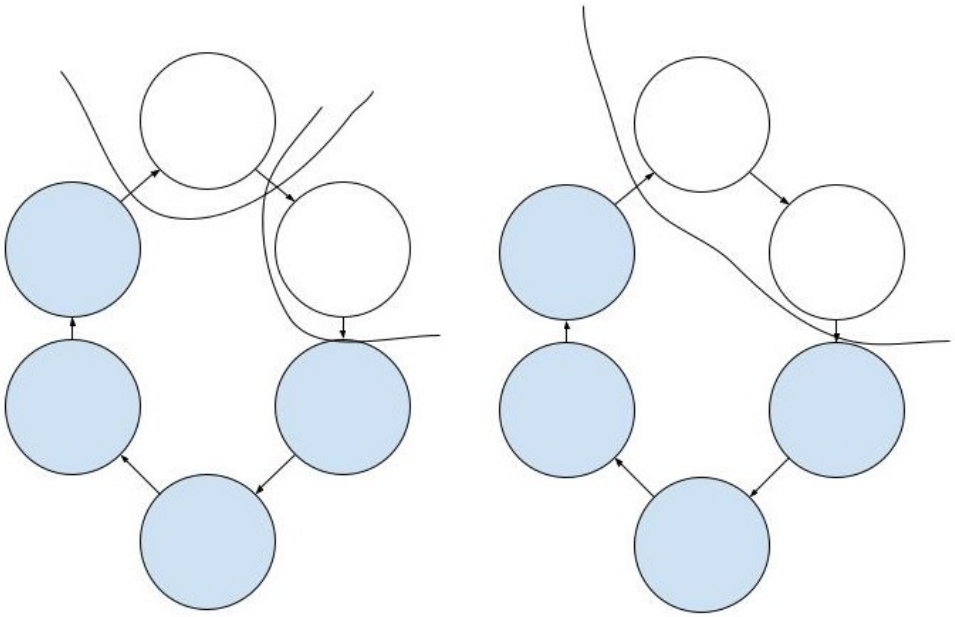

FIGURE 1.12: $Decider < x^6/x^2 >$

$Decider < x^6/x^1/x^1 >$ $Decider < x^6/x^2 >$

Determining if y is in f x is easy if we are given any monomial decider in the set of the language of polynomials and their representations has the possibility to give different representations if we consider them as representations of the function f of x.

$Decider < x^6/x^1/x^1 > \sim Decider < x^6/x^2 >$ in that they decide if y is in m(x) = x^6/Q

Theorem of Equivalence. Something on lines of $Decider < x^6/x^1/x^1 >= Decider < x^6/x^2 >$ such that there is some x such that the monomial represented by both deciders exists where f of x = y.

Proof. Given $Decider < a(x,y) >= Decider < x^m > Decider < y^n >$ and the definition $Decider < b(x,y) >= Decider < y^n > Decider < x^m >$, show $Decider < a(x,y) >= Decider < b(x,y) >$.

Start with $Decider < a(x,y) >= Decider < x^m > Decider < y^n >$
Commute the variables because each decider decides there own monomial giving, $Decider < y^n > Decider < x^m >$.

This is equivalent to $Decider < b(x,y) >$

Start with $Decider < b(x,y) >= Decider < y^m > Decider < x^n >$
Commute the variables because each decider decides there own monomial giving, $Decider < x^n > Decider < y^m >$.
This is equivalent to $Decider < a(x,y) >$

1.14 Reversing

$Decider < x^6/x^1/x^1 >\equiv$sequence of permutations such that it is equal to $\sum_i^{n-1} i$

$Decider < x^6/x^2 >\equiv$ sequence of permutations of x_i, x_j such that it equals n-1.

Is shown that by the permutation of the order of operations that $Decider < x^6/x^1/x^1 >$ does not have the same number of permutations as $Decider < x^6/x^2 >$

Theorem of Reversing. Given two representations, a,b in $Decider < m(x)/Q >$ where m(x) is monomial and Q is the division operations such that m(x)/Q $\geq$ 1, a != b implies that they don't have the same quotients space.

Proof. Given $Decider < a(x,y) >= Decider < x^m > Decider < y^n >$ and the definition $Decider < b(x,y) >= Decider < y^n > Decider < x^m >$, show $Decider < a(x,y) >\neq Decider < b(x,y) >$.

Start with $Decider < a(x,y) >= Decider < x^m > Decider < y^n >$.
Then take $Decider < x^m y^n >$.
Commute to get $Decider < y^n x^m >$.
This is not equivalent to $Decider < y^n > Decider < x^m >= Decider < b(x,y) >$.

Start with $Decider < b(x,y) >= Decider < y^n > Decider < x^m >$.
Then take $Decider < y^n x^m >$
Commute to get $Decider < x^m y^n >$
This is not equivalent to $Decider < x^m > Decider < y^n >= Decider < a(x,y) >$

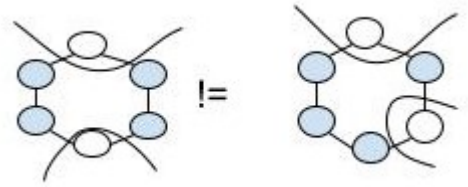

FIGURE 1.13: Two possible representations of *Decider* $< x^6/x/x >$.

The two representations in 1.13 don't have the same quotient space and hence a != b.

1.15 Corollary of Reversing

Given a starting point of decider, the path the decider takes to decide if y is in the monomial, m(x) is unique to each representation.

Corollary. Given a decider, d, in *Decider* $< m(x) >$ then there is path, p, that exists for d such that p = Path(d) = $s_1, s_2, ..., s_i, ..., s_n$ where i is count of the states in the decider of m(x).

Example. Choose some x such that it is in path of *Decider* $< x^6/x^2 >$ where $p = 001111$ then the following graph is what the decider is represented as.

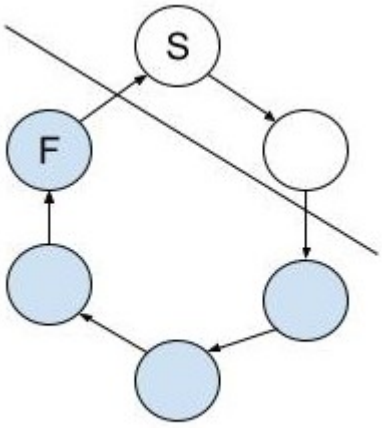

FIGURE 1.14: Path of one representation of *Decider* $< x^6/x^2 >$.

1.16 Godel's Theorem

We see that there exists two statements from these theorems

1. $x = x$ from a theorem of equivalence
2. $x! = x$ from a theorem of reversal

Example: Given some d_1,d_2,d_3,...,infinity in decision functions in $Decider < m(x) >$

1. $d_1 = d_2 = d_3 = ... = infinity$
2. $d_1! = d_2! = d_3 = ...! = infinity$

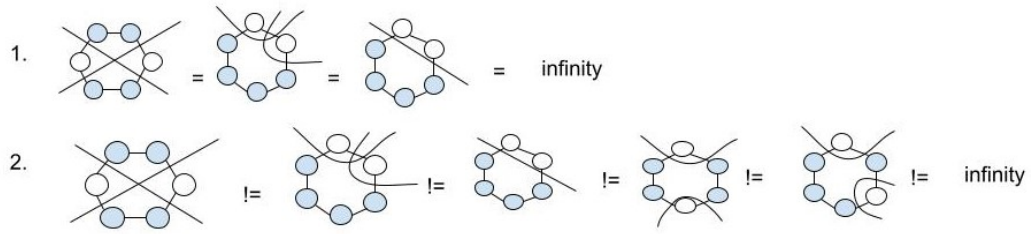

FIGURE 1.15: Godel's illustrated from $Decider < x^6/x^2 >$.

The different representations of a monomial through the language of monomial deciders will give the problem of undecidability. This means that despite many formal definitions of the monomial decider, there is no way to solve the problem of finding a specific representation of a monomial decider without having to guess or apply some sort of probability to it. Relating to the real line, given a real line a,b a $\leq$ b, there is infinite choices between a and b. As long as b and a ≥ 0, there requires some sort of probability of choosing some specific number that is between a and b.

1.17 Constructing The One Way Function

A probability exists to find a certain monomial decider in the set of it's variations. A/B = Probability where A is the monomial decider we want and B is the number of all the variations.

Example: d_1,d_2,...,d_6 in $Decider < x^6 >$ such that d_i are all distinct.
Choose one of the deciders in D through probability
Probability of choosing d in D is 1/6 so 0.16666667

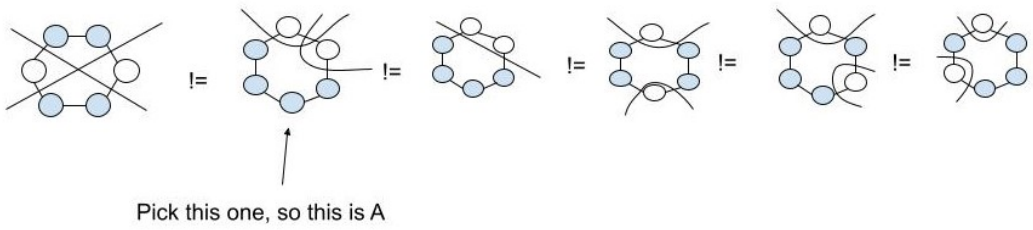

FIGURE 1.16: Picking a decision function, d, from $Decider < x^6/x^2 >$.

This is called the picking function and every time it is called, the probability is multiplied such that it is n^k. As an example, if the picking function is called twice using the example above, it is shown that the probability is:
$1/6 \times 1/6 = 1/6^2 = 1/36 = 02777778$

This is formally known as the one way function.

Chapter 2

Analysis of Fibonacci

2.1 Euler's Constant

From the the reinterpretation of the theory, Euler's constant is an example of P = NP because of it's use of calculus. Euler's constant can be defined as

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

To show that it is also in the problem set of P $\neq$ NP, start by using the picking function going into infinity. The constant e is the sum of the infinite series of $\frac{1}{n!}$ and can be represented as a series of monomials representing the Decider<x> using the picking function.

$$e = (\sum_{n=0}^{\infty} \frac{1}{n!})$$

$$e = 1 + 1 + 1/(1+1) + 1/(3+3) + 1/(4 + (4*3 + 4*2) + 4) + ...$$

$$e = p_f(Decider < x^0 >) + p_f(Decider < x >) + p_f(Decider < x^2/x > + ...$$

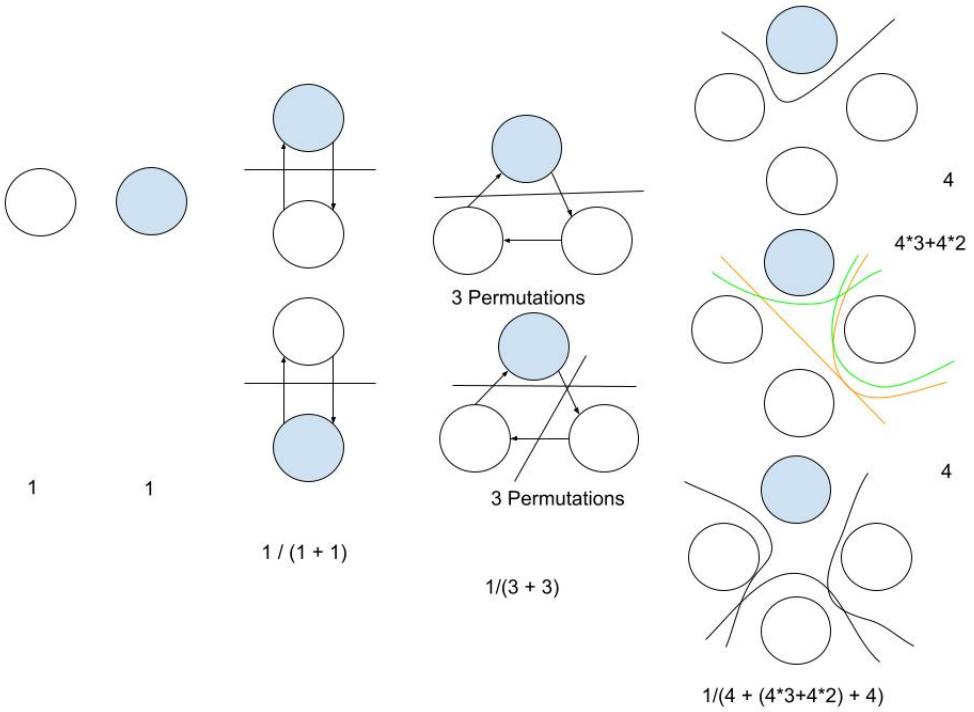

FIGURE 2.1: Decider that represents the first four terms of the constant e.

2.2 Example of a Decider

The following is an example of code that roughly sketches a generalized monomial decider, Decider<$2x^2$>. It tests to see if y is in the monomial m(x) = $2x^2$. Although approximation and binary search algorithms work, using this method allows for simplicity of technique in representing a decider visually and gives the reader hands-on potential if they want to experiment on the subject further.

```
1:  procedure GENERALIZEDDECIDER(y)
2:      if y = 0 then return true
3:      end if
4:      (s ← y)
5:      while s ≥ 0 do
6:          for i = 0 to 1 do
7:              for j = 0 to 3 do
8:                  if s = 0 and (j = 0 or j = 2) and i = 0 then return true
9:                  else if s < 0 then return false
10:                 end if
11:                 s ← s − 1
12:             end for
13:         end for
14:     end while
15: return true
16: end procedure
```

Proof of Correctness.

Initialization. On lines 2 to 3, if y is 0 then it is true. Otherwise, take s to be y.

Maintenance. The loop invariant starts at line 5 as $s \geq 0$ with value of s being y. While this is true it goes through a for loop on lines 6 to 13 with a starting value of i at 0. At line 7, the last for loop starts with j being a value of 0. On line 8, a condition is checked to see if s is 0, i is 0, and j is either 0 or 2 and returns true if it is. It then passes the for loop with j as the iterating variable until it hits 3 and then the for loop with i as the iterating variable until it hits 1. It continues another pass of the while loop.

Termination. The while loop on lines 5 to 14 terminate if the for loop on lines 6 to 13 terminate. This for loop terminates if the for loop inside it on lines 7 to 12 terminate. s decreases by 1 on line 11 while once it passes the condition checks and it goes through it at most 2 * 4 times representing the passes in the nested for loop. This algorithm always either returns true or false.

Proof of Time Complexity. Starting from the inner most for loop on lines 7 to 12 iterate 3 times. Outside is the for loop on lines 6 to 13 that iterates 2 times. This gives a time complexity of exactly 6 constant. The while loop on lines 5 to 14 has $s \geq 0$ initially s = y which gives a time complexity of $O(y)$. Overall, this gives a time complexity of $O(y + 6) = O(y)$.

Proof of Space Complexity. There is only one variable used to keep track of the iteration and that is the variable, s. There are the iteration variables i and j. Hence, the algorithm has a space complexity of a constant, $O(3) = O(1)$.

The above code can be represented visually as follow:

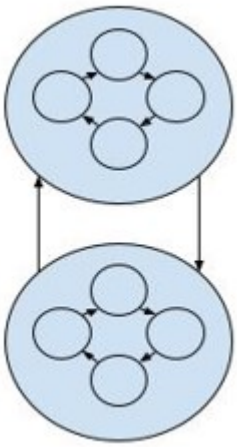

FIGURE 2.2: Decider that represents the monomial, x^3.

2.3 Analysis of the Decider $2x^2$

Data can be extracted to find insight and build a general technique using the generalizedDecider function above for the Decider<$2x^2$>.

Generate	Even Input	Even Output
f(0) = 0	0	1
f(1) = 2	1	1
f(2) = 8	0	1
f(3) = 18	1	1
f(4) = 32	0	1
f(5) = 50	1	1
f(6) = 72	0	1
f(7) = 98	1	1
f(8) = 128	0	1
f(9) = 162	1	1
f(10) = 200	0	1
f(11) = 242	1	1
f(12) = 288	0	1
f(13) = 338	1	1

FIGURE 2.3: Generate function, $2x^2$.

Generate is f(x) = $2x^2$ = y on the first line and the number of negatives is on the second. There are two finishing and all the even parity outputs end in one state and

all the odd parity outputs end on the other. **Even Input** is the boolean parity of the input being even. **Even Output** is the boolean parity of the output being odd.

Here, it can be seen that there is repeating pattern of even and odd values of the following:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

From this array, two finishing states are described. When the input and output are both even and one where the input is odd and the output is even. Every time the head of the tape passes a finishing state, the count of the variable, **visit** is increased by one. Counting the number of visits two the finishing states, it can be seen that for every pair, (start,end), there is a difference of three for $Decider < 2x^2 >$. The following is a table of the results:

Generate	Visits	Difference
f(0) = 0	0	0
f(1) = 2	1	1
f(2) = 8	4	3
f(3) = 18	9	5
f(4) = 32	16	7
f(5) = 50	25	9
f(6) = 72	37	11
f(7) = 98	50	13

FIGURE 2.4: Analysis of visits to finishing states in, $2Decider < 2x^2 >$.

From analysis, the number of times the tape head visits a finishing state is the difference of the previous input plus two. The difference increases by two every time it passes a finishing state. It doesn't pass when the number of visits is even, so it can be deduced that the difference is increased by two every time it passes the odd finishing state. This result allows us to write a program to decide if y is in $2x^2$.

```
1:  procedure GENERATOR(max)
2:      result ← [ ]
3:      x ← 0
4:      difference ← 0
5:      i ← 0
6:      while x < max + 1 do
7:          num ← 2x^2
8:          if GeneralizedDecider(i) then
9:              if num = i then
10:                 result[x] = i
11:                 x ← x + 1
12:                 difference ← 0
```

```
13:             else
14:                 difference ← difference − 1
15:             end if
16:         end if
17:         i ← i + 1
18:     end while
19: return result
20: end procedure
```

Proof of Correctness.

Initialization. The variables result, x, difference, and i get initiated.

Maintenance. The loop is invariant with the case $x < max + 1$ which holds true. The num variable is set to $2x^2$. $GeneralizedDecider(i)$ is used to decide if i is in the decider, but it isn't always in $2x^2$. If num is i, the variable x is updated to keep the condition of the loop invariant true which is $x < max + 1$. On line 17, the variable i is updated so that on line 8, the condition is always met.

Termination. Start of with the while loop such that $x < max + 1$. i is incremented every loop so that the condition $GeneralizedDecider(i)$ is always met on line 8. This means that when it visits line 9, the condition is passed and the variable, x, is increased moving the while loop on line 6 forward. This allows the algorithm to terminate successfully.

Proof of Time Complexity. The time complexity of the algorithm starts with line 6 of the while loop. $x < max + 1$ is the condition that meets to be met in order for it to terminate and the variable, x, is set to zero on line 3. On line 8, the call to *GeneralizedDecider* is made every time it loops - this has a time complexity of i. Coming back to line 6, this loop has a time complexity of $O(n)$ where n is equal to max. Concluding this analysis, the summary of the overall time complexity is $O(i * n) < O(n^2)$ where i is 0 to n.

Proof of Space Complexity. The space complexity of the algorithm has three constant variables - x, $difference$, and i. The array *result* has a size of n where n is max. Hence, the space complexity is $O(n + 1 + 1 + 1) = O(n)$

2.4 Representing Monomial Deciders As Code

With the data above, the requirements on constructing a decider is as follows.

Given the function:

$f(x) = 2x^2 = \{0, 2, 8, 18, \cdots\}$

The output of f(x) is of the following.
x is even at 0,8,32,72
x is odd at 2,18,50

There are four variables constructed:
Current passes records the number of times path traveled passes a finishing state.
Total number of times traveled on a finishing state needed to reach a valid decision.
Diff is the current number of diff to increment total hits by.
IsEven is if this resets back to even, increment Diff by two.

The following is code generated from our more formal representation of the solution.

```
1:  procedure DECIDER(y)
2:      totalVisits ← 0
3:      currentVisits ← 0
4:      difference ← 1
5:      s ← 0
6:      while s ≤ y do
7:          for i = 0 to 1 do
8:              for j = 0 to 1 do
9:                  for k = 0 to 1 do
10:                     if i = 0 and (j = 0 or j = 1) and k = 0 then
11:                         if currentVisits = totalVisits then
12:                             if s = y then return true
13:                             else if s > y then return false
14:                             end if
15:                             totalVisits ← totalVists + difference
16:                             if i = 0 and j = 1 and k = 0 then
17:                                 difference ← difference + 2
18:                             end if
19:                             currentVisits ← currentVisits + 1
20:                         end if
21:                         s ← s + 1
22:                     end if
23:                 end for
24:             end for
25:         end for
26:     end while
        return false
27: end procedure
```

Proof of Correctness.

Initialization. The variables totalVisits, currentVisits, difference, and s are all initialized with values. The iterator variable $i \leftarrow 0$ is set to 0 so that it is ready for the loop invariant to to be used.

Maintenance. The loop is invariant has the condition $s \leq y$ which holds true because s is initialized with a value of 0. The for loops between lines 7 to 8 are iterated through. The condition on line 10 is true if the head visits a travels on an accept state. On line 15, $totalVisits$ is increased by the difference and difference is increased by two if it hits the accepting state that accepts odd traversals. $currentVisits$ is increased by one which means that it holds up to the condition $currentVisits = totalVisits$. Finally, line 21 updates the variable, s, so that it moves the loop invariant forward.

Termination. Start with s set to 0. Line 21 implies that s is always incremented despite the conditional on line 11. It increments when the head visits line 10. When the loop invariant condition is false or when $s > y$, it means that the algorithm terminates. The algorithm also terminates if $s = y$ on line 12 and on line 13 when $s > y$ is true.

Proof of Time Complexity. The condition on line 6, $s \leq y$, shows that s goes through values 0 to y. Between lines 7 to 9, i, j, and k loop $2^3 = 8$ times which might appear constant, however, the condition, $currentVisits = totalVisits$ means that there is a total amount of $totalVisits$ visits to the for loop invariant. This means there are $O(totalVisits * totalVisits) < O(n^2)$ time complexity.

Proof of Space Complexity. From initialization, there are four variables with values. They take up a space, $O(1 + 1 + 1 + 1) = O(1)$ space.

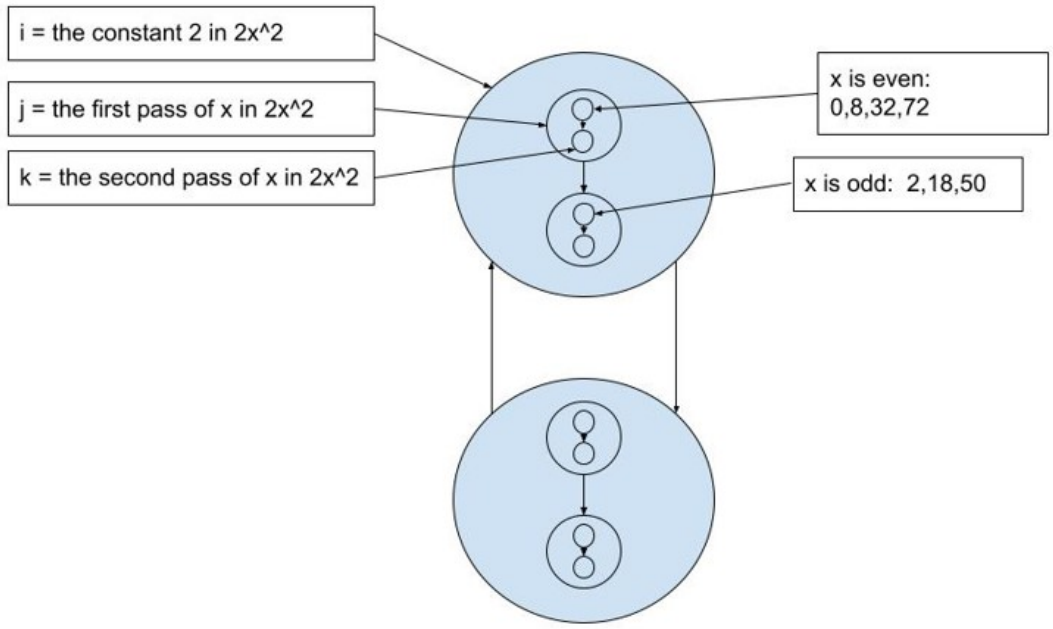

FIGURE 2.5: The algorithm described visually, $2Decider < 2x^2 >$.

2.5 Negative Monomials

Representing negative numbers can be thought of discretely. Below is a representation of negative numbers.

Addition of two negative deciders gives a negative decider.
$Decider < -x > + Decider < -x >$
$= Decider < -x - x >$
$= Decider < -2x >$

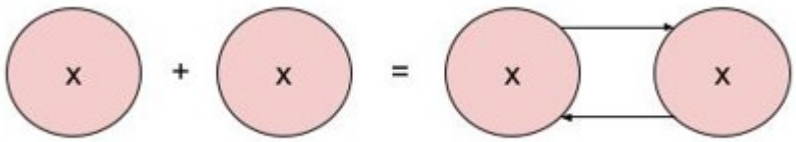

FIGURE 2.6: Addition.

Cancellation of a positive and negative decider of such that it is the additive inverse, or 0.

$Decider < x > + Decider < -x >$
$= Decider < x - x >$
$= Decider < 0 >$

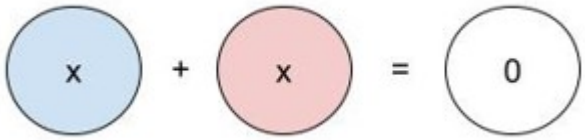

FIGURE 2.7: Cancellation law.

Multiplication of two negative deciders gives a positive decider.

$$Decider < -x > * Decider < -x >$$
$$= Decider < -x * -x >$$
$$= Decider < x^2 >$$

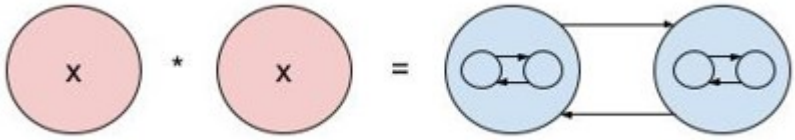

FIGURE 2.8: Multiplication.

Multiplication of a negative decider with its multiplicative inverse gives a its identity, or 1.

$$Decider < x^{-1} > * Decider < x >$$
$$= Decider < x^{-1} * x >$$
$$= Decider < 1 >$$

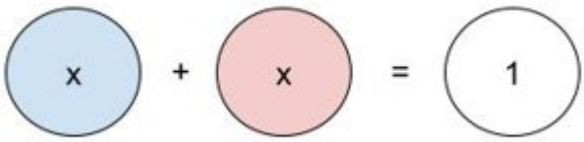

FIGURE 2.9: Multiplicative Inverse.

2.6 Pi

Representing the constant pi, π, in the language of polynomials using the Leibniz formula $\pi/4 = 1 - 1/2 + 1/5 - 1/7 + 1/9 + \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$

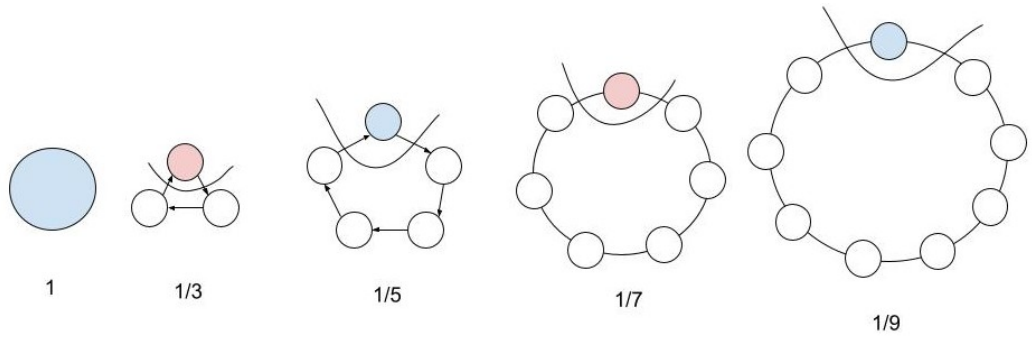

FIGURE 2.10: Pi under the Leibniz formula to illustrate choosing one state in a decision function of a term decider.

2.7 Analysis of Fibonacci

A Fibonacci sequence is a sequence of typically seen as the following, $1, 1, 2, 3, 5, 8, 13, 21, \cdots$. The general formula for this sequence is:

$a_n = a_{n-1} + a_{n-2}$ given a_0 and a_1

From the example above, we see that f(1) = 1 and f(2) = 2. If we add f(1) and f(2) together we get f(3) = 3 and so on and so forth. The algorithm below will be used to collect data to be analyzed to find a general pattern:

```
1:  procedure FIBONACCI(n)
2:      if n = 1 then return 0
3:      end if
4:      y ← 1
5:      y1 ← 1
6:      y2 ← 2
7:      for  doi = 1ton − 1
8:          y ← y1 + y2
9:          y1 ← y1
10:         y2 ← y
11:     end for
          return y
12: end procedure
```

The data collected is organized into input, its parity, output, and its parity. First, the input and its parity value is analyzed in the following:

Input	Parity of Input
1	0
2	1
3	0
4	1
5	0
6	1
7	0
8	1

FIGURE 2.11: The input and the parity of the input of Fibonacci.

The parity alternates between 0 and 1 which doesn't mean much on its own. Collecting data from the output of the sequence function gives the following:

Output	Parity of Output
1	0
1	0
2	1
3	0
5	0
8	1
13	0
21	0

FIGURE 2.12: The output and the parity of the output of Fibonacci.

On analysis, it can be seen that there are two patterns mapped out - one from input values 1, 2, and 3 (called 123) and one from input values 4, 5, and 6 (called 456). Laying these findings flat on a 3×3 matrix gives the following:

{123}	{456}	Y
0	1	0
1	0	0
0	1	1

FIGURE 2.13: The output Y is the output parity.

There are two matrices that form from analysis of the parity of the input and output. Using the technique to develop an algorithm for the *Decider* $< 2x^2 >$ it can be concluded that there are three finishing states for each matrix giving a total of six different finishing states.

2.8 Modeling Deciders of Fibonacci

There exists a mapping between the determinants of the Fibonacci sequence to each state of the Fibonacci sequence. On analyzing the parity of the input of 123, 456 and the output of the fibonacci sequence, two matrices emerge. Label the input 123 as E_1, the input of 456 as E_2, and the output of the Fibonacci sequence as E_3.

E_1	E_2	E_3
0	1	0
1	0	1
0	1	1

There are six permutations pivoting by the row of the matrices. Three of them have a determinant of -1 and the other three have a determinant 1. The three matrices that have the determinant of -1 are as follows.

$$M_1 = \begin{array}{ccc} E_1 & E_2 & E_3 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}$$

$$M_2 = \begin{array}{ccc} E_1 & E_2 & E_3 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{array}$$

$$M_3 = \begin{array}{ccc} E_1 & E_2 & E_3 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{array}$$

The next three matrices that have a determinant of 1 is as follows.

$$M_4 = \begin{array}{ccc} E_1 & E_2 & E_3 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}$$

$$M_5 = \begin{array}{ccc} E_1 & E_2 & E_3 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{array}$$

$$M_6 = \begin{array}{ccc} E_1 & E_2 & E_3 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{array}$$

These matrices above would otherwise be called the modulo inverse in most format because it takes the parity, or modulo, of the input and output.

In order to model the Fibonacci sequence, swap the rows E_1 and E_2 of M_1 resulting in the matrix N_1. The determinant of N_1 is 1 and the rows still have integrity. The matrices can be turned into a sort of adjacency matrix and mapping the the entries of E_1 of M_1 and E_2 of N_1 to the value of the states can be seen as a graph in the following:

$E_1 \equiv -x = y - z$
Determinant of M_1 = -1

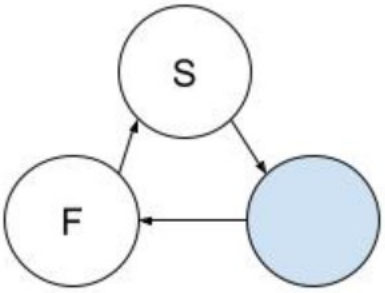

FIGURE 2.14: The determinant of input parity E_1 of M_1 modeled as a graph visually.

$E_2 \equiv x = -y + z$
Determinant of N_1 = 1

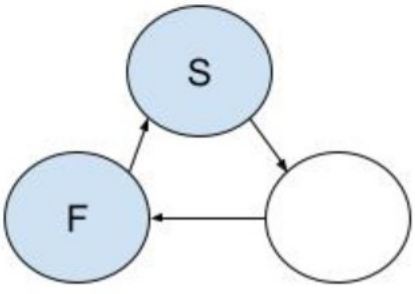

FIGURE 2.15: The determinant of input parity E_2 of N_1 modeled as a graph visually.

After higher level illustrations and, in addition to the matrices, there are representations of the sequences and, more generally, words that can be formed if a more detailed view of how it operates is needed in linear algebra. A definition below can be traced.

Definition. $(S, w) = \lambda\mu w\gamma$.

Set λ to a row vector of length 3 filled with ones and γ and operate on the matrix, M_1 and N_1.

$$\lambda\mu_{101} = \begin{bmatrix}1 & 1 & 1\end{bmatrix} \begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 0\\ 0 & 1 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 1\end{bmatrix}$$

$$\lambda\mu_{011} = \begin{bmatrix}1 & 1 & 1\end{bmatrix} \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 1 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 1 & 1\end{bmatrix}$$

This result means that there is one even column that permutes between M_1 and N_1. Now γ can be described as the rows being permuted which is the same for both matrices.

$$\gamma = \begin{bmatrix}1\\ 1\\ 0\end{bmatrix}$$

After representing the Fibonacci sequence as matrices, the following theorem below can be reached.

Theorem. A formal series is regular if and only if it is rational.

Proof. Here, regular and recognizable are equivalent terms and a regular language is a subset of a decider. A monomial decider is a variant of a decider.

Taking the sum of the path of the graph and mapping the state's that represent 0 to -1, the following can be seen.

$E_1 \equiv -x = y - z$
Path of M_1 = -1 + 1 - 1 = -1

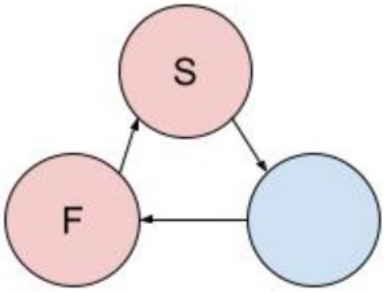

FIGURE 2.16: The sum of input parity E_1 of M_1 modeled as a graph visually.

$E_2 \equiv x = -y + z$
Path of N_1 = 1 - 1 + 1 = 1

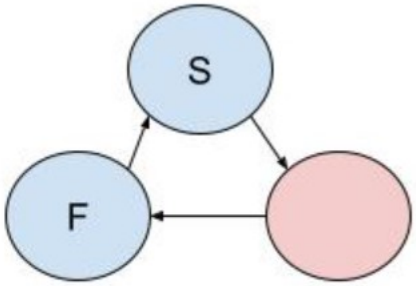

FIGURE 2.17: The sum of input parity E_2 of N_1 modeled as a graph visually.

2.9 Redrawing the Fibonacci Sequence

From our analysis, it can be seen that there are three states that are a minimum to create a Fibonacci sequence. Minimization gives us a monomial generator, S_x, S_y, S_z. Set the three states to a desired configuration (i.e. $S_y = 0$ and $S_z = 1$ and it will generate the Fibonacci sequence. It can shown that it requires three states minimum to generate the Fibonacci sequence.

Generator
$S_x = S_y + S_z$
$S_y = S_z + S_x$
$S_z = S_x + S_y$

Using the above, dynamic programming can be modeled as a set of generator functions. A visual diagram explains how a set of generator functions form a generator.

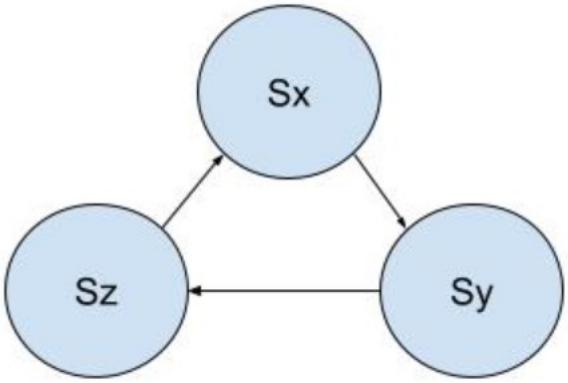

FIGURE 2.18: The generator for the Fibonacci sequence.

```
int fibonacciGenerator(int n)
{
    int stateX = 0;
    int stateY = 1;
    int stateZ = 1;
    int cycles = 0;

    while (cycles <= n)
    {
        cycles++;

        if (cycles > n)
        {
            return stateX;
        }

        stateX = stateY + stateZ;
        Console.WriteLine("stateX: " + stateX + "\tstateY: " + stateY + "\tsta

        cycles++;

        if (cycles > n)
        {
            return stateY;
        }

        stateY = stateX + stateZ;
        Console.WriteLine("stateX: " + stateX + "\tstateY: " + stateY + "\tsta
```

```
            cycles++;

            if (cycles > n)
            {
                return stateZ;
            }

            stateZ = stateX + stateY;
            Console.WriteLine("stateX: " + stateX + "\tstateY: " + stateY + "\t
        }

        return stateX;
    }
```

2.10 The Fibonacci Decider

Given the two deciders we found using the determinant and the sum of the path by the mapping of the determinants, it is shown that six decision functions and a minimum of six input values are required to decide if a sequence is a type of Fibonacci sequence. In order to create this decider, a composition operator is used on M_1 and N_1 to merge them together because the sum of the determinant cancels each other. All decision functions must evaluate to true in order for the sequence to be a valid Fibonacci sequence. The following is one possible set of permutations available to form a group of decision functions with the deciders.

Example 2.11.1. Decider with Decision Functions for {123}
$x_1 = -y_1 + z_1$
$-x_1 = y_1 + z_1$
$x_1 = y_1 - z_1$

Example 2.11.2. Decider with Decision Functions for {456}
$-x_2 = y_2 + -z_2$
$x_2 = -y_2 - z_2$
$-x_2 = -y_2 + z_2$

Join the finish state of {123} to the start state of {456} and the start state of {456} to the finish state of {123} to form the Fibonacci decider {123456}

Example 2.11.3. Fibonacci decider {123456}
$x_1 = -y_1 + z_2 - x_2 + y_2 - z_1$
$-y_1 = z_2 - x_2 + y_2 - z_1 + x_1$
$z_2 = x_2 + y_2 - z_1 + x_1 - y_1$
$x_2 = y_2 - z_1 + x_1 - y_1 + z_2$
$y_2 = z_1 + x_1 - y_1 + z_2 - x_2$

$z_1 = x_1 - y_1 + z_2 - x_2 + y_2$

2.11 The Fibonacci Picking Function

On analysis, one decision function in $\{123\}$ has three possible permutations. An example is of the following.

Given $x_1 = -y_1 + z_1$, it has $x_1 + y_1 = z_1$ and $x_1 - z_1 = -y_1$.

This means that every decision function in $\{123\}$ has three possible permutations, or 3^2 possibilities, however, if the configuration is desired to be M_1 and N_1, then there is only one configuration possible out of 3^2 by 3^2 possibilities.

After, it can be shown that by applying the picking function, p_f, to the Fibonacci decider, the probability is equivalent to $1/n^k$.

Each decider has three representations giving $3^2 = 9$ total for each set. There is only one configuration of M_1 and N_1 given to imply that the probability is $1/9$. Multiply the probability of picking M_1 by the probability of picking N_1 to get $1/9^2$.

There are 6 deciders with 6 permutations each and giving $6^2 = 36$ permutations.

The probability is then:

$1/3^2 * 1/3^2 \leq 1/6^2$

$\equiv p_f(\{123\}) * p_f(\{456\}) \leq 1/n^k$ where k = 2

By definition this is called the one way function.

This shows that the relation between $M_1\ of\ \{123\}$ and $N_1\ of\ \{456\}$ and the formation of $\{123456\}$.

This shows a concrete example of the picking function which is a type of one way function.

Chapter 3

Inferrable Languages

3.1 Introduction

The concept of statistics and blackboxes has been drawn out extensively in theories and applications for decades but what of languages and knowing what word can be used to generate the next series of words? Everyone guesses what words can come out of someone talking given enough experience. In this article, the idea of inferrable languages is presented which are languages that allow the next series of words in the sequence to be inferred given enough samples in the sequence.

3.2 Applying The Fibonnaci Decider

Given the definition of the Fibonacci decider and a Lindenmayer system, insight can be derived from to that there exists the commutative and noncommutative properties of the operations.

Decider for x_1, y_1, z_1
$-x_1 = y_1 - z_1$
$y_1 = -x_1 - z_1$
$-z_1 = x_1 - y_1$

Decider for x_2, y_2, z_2
$x_2 = y_2 - z_2$
$-y_2 = x_2 - z_2$
$z_2 = x_2 - y_2$

Fibonacci Decider
$x_1 = -y_1 + z_2 - x_2 + y_2 - z_1$
$-y_1 = z_2 - x_2 + y_2 - z_1 + x_1$
$z_2 = -x_2 + y_2 - z_1 + x_1 - y_1$
$-x_2 = y_2 - z_1 + x_1 - y_1 + z2$
$y_2 = -z_1 + x_1 - y_1 + z_2 - x_2$
$-z_1 = x_1 - y_1 + z_2 - x_2 + y_2$

Lindenmayer System
L is the definition of the D0L System

$L = (V, \omega, P)$
V are the characters in the language called the alphabet
ω is the starting string called the start
P are the production rules in the language called the rules

$alphabet. = \{a, b\}$
$rules. = \{a \Rightarrow ab, b \Rightarrow a\}$
$start. = b$

3.3 Fibonacci DOL Decider Left Hand Side

Representing the left hand side of the Fibonacci sequence as a D0L system alphabet requires an alphabet, rules, and a starting state.

The first six sequences in the

b a ab aba abaab abaababa abaababaabaab abaababaabaababaabaab

The length are as follows.

1 1 2 3 5 8 13 21

Set them as variables in the decider as follows.

Decider for x_1, y_1, z_1
$-x_1 = y_1 - z_1$
$y_1 = -x_1 - z_1$
$-z_1 = x_1 - y_1$

Decider for x_2, y_2, z_2
$x_2 = y_2 - z_2$
$-y_2 = x_2 - z_2$
$z_2 = x_2 - y_2$

$z_2 = b$
$y_2 = a$
$x_2 = ab$
$z_1 = aba$
$y_1 = abaab$

$x_1 = abaababa$

Decider for x_1, y_1, z_1

$-x_1 = y_1 - z_1$
$-\ abaababa = abaab - aba$
$-abaababa + aba = abaab$
$-abaababa - abaab = -aba$

$y_1 = -x_1 - z_1$
abaab = - abaababa - aba
abaab + abaababa = - aba
abaab + aba = - abaababa

$-z_1 = x_1 - y_1$
- aba = abaababa - abaab
- aba + abaab = abaababa
- aba - abaababa = - abaab

Decider for x_2, y_2, z_2
$x_2 = y_2 - z_2$
$ab - b = a$
$ab - a = b$

$-y_2 = x_2 - z_2$
-a + b = ab
-a - ab = -b

$z_2 = x_2 - y_2$
b - ab = -a
b + a = ab

Generalized equations for x_1, y_1, z_1

$-a = b - c$
$-a + c = b$
$-a - b = -c$

b = -a - c
b + a = -c
b + c = a

-c = a - b
-c + b = a
-c - a = -b

Generalized equations for x_2, y_2, z_2

a* = *b* + *c
$a - c = b$
$a - b = c$

b = a - c
b - a = -c
b + c = a

-c = a - b
-c + b = a
-c - a = -b

3.4 Fibonacci DOL Decider Right Hand Side

Fibonacci Decider

$x_1 = -y_1 + z_2 - x_2 + y_2 - z_1$
$-y_1 = z_2 - x_2 + y_2 - z_1 + x_1$
$z_2 = -x_2 + y_2 - z_1 + x_1 - y_1$
$-x_2 = y_2 - z_1 + x_1 - y_1 + z2$
$y_2 = -z_1 + x_1 - y_1 + z_2 - x_2$
$-z_1 = x_1 - y_1 + z_2 - x_2 + y_2$

abaababa = -abaab + b - ab + a -aba
$abaababa - b + ab - a + aba = -abaab$
$abaababa + abaab + ab - a + aba = b$
$abaababa + abaab - b - a + aba = -ab$
$abaababa + abaab - b + ab + aba = a$
$abaababa + abaab - b + ab - a = -aba$

-abaab = b - ab + a -aba + abaababa
$-abaab + ab - a + aba - abaababa = b$
$-abaab - b - a + aba - abaababa = -ab$
$-abaab - b + ab + aba - abaababa = a$
$-abaab - b + ab - a - abaababa = -aba$
$-abaab - b + ab - a + aba = abaababa$

b = -ab + a - aba + abaababa - abaab

$$b - a + aba - abaababa + abaab = -ab$$
$$b + ab + aba - abaababa + abaab = a$$
$$b + ab + aba - abaababa + abaab = -aba$$
$$b - ab + a - aba + abaab = abaababa$$
$$b - ab - a + aba - abaababa = -abaab$$

-ab = a - aba + abaababa - abaab + b

$$-ab + aba - abaababa + abaab - b = a$$
$$-ab - a - abaababa + abaab - b = -aba$$
$$-ab - a + aba + abaab - b = abaababa$$
$$-ab - a + aba - abaababa - b = -abaab$$
$$-ab - a + aba - abaababa + abaab = b$$

a = aba + abaababa - abaab + b - ab

$$a + abaababa - abaab + b - ab = aba$$
$$a - aba - abaab + b - ab = -abaababa$$
$$a - aba + abaababa + b - ab = abaab$$
$$a - aba + abaababa - abaab - ab = -b$$
$$a - aba + abaababa - abaab + b = ab$$

-aba = abaababa - abaab + b - ab + a

$$-aba + abaab - b + ab - a = abaababa$$
$$-aba - abaababa - b + ab + a = -abaab$$
$$-aba - abaababa + abaab + ab - a = b$$
$$-aba - abaababa + abaab - b - a = -ab$$
$$-aba - abaababa + abaab - b + ab = a$$

3.5 The Law of Commutativity and Noncommutativity

The law of commutativity and the law of noncommutativity combined gives the law of commutativity and noncommutativity

The Law of Commutativity
a + b = b + a
ex. 8 + 5 = 5 + 8

The Law of Noncommutativity
a+ b != b + a
ex. 8 - 5 != 5 - 8

Each equation in the example on the left has permutations.

From this example, it can be implied that for every variable, n, in an equation there is n^2 permutations in the sequence.

The first equation is bold and italicized in the set to make a decider.

a = b + c	**b = a - c**	**-c = a - b**
$b = a - c$	$b - a = -c$	$-c + b = a$
$-c = a - b$	$b + c = a$	$-c - a = -b$

3.6 Operations

Operations for the right hand side (RHS) versus the left hand side (LHS) represents different operations of the string in different scenarios.

RHS Evaluation
Right to Left
+ Remove from the back
- Add to the front
abaababa = - abaab + b - ab + a -aba
abaababa = - abaab + b - ab -ab
abaababa = - abaab + b - abab
abaababa = - abaab - aba
abaababa = - abaababa

LHS Evaluation
Left to Right
+ Remove at the front
- Add to the back
abaababa + abaab - b + ab - a = -aba
aba - b + ab - a = -aba
abab + ab - a = -aba
ab - a = -aba
aba = -aba

*Proposition.*The characteristic series of a rational cyclic language is a Z-linear combination of characters of finite deterministic automata.

3.7 Definition Of Support

A support is defined as the following:
A* is a word
S is the function
Image by S of a word w is denoted by (S,w) and is the coefficient of w in S
Support(S) = w in A* such that (S,w) != 0

Now we take deciders of a monomial and the picking function to redefine the support of a noncommutative rational language
R is the rational numbers where x in R = a/b
such that a,b is in integers and b $\neq$ 0

Q is the quotient space represented in topology such that A/ where are sets and is the divisions of A

R-rational is the representation of R as the polynomial function

Q-rational is the representation of Q as a polynomial

3.8 Rationals Of Picking Function

Support of the Fibonacci Picking Function of the deciders of a monomial.

Q-rational-deciders are the possible monomial deciders of Q-rational-string. Use the picking function, PF, to choose one decision function in Q-rational-deciders, we see a mapping from Q-rational => R-rational.

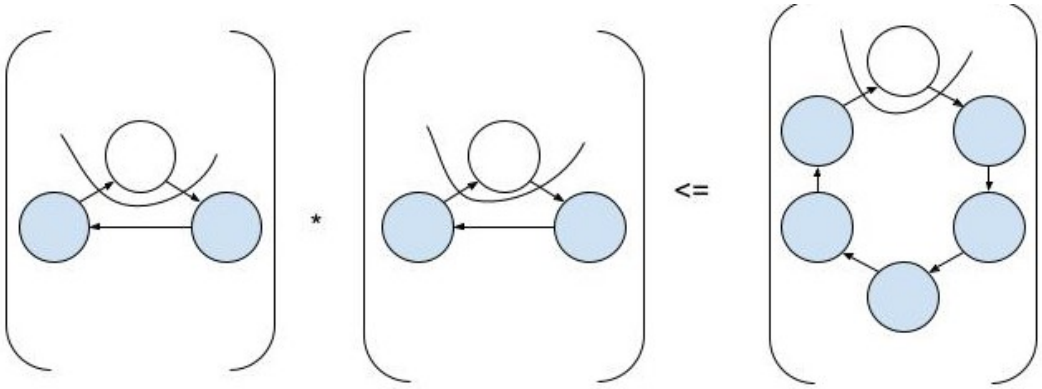

FIGURE 3.1: Q rational image of the q rational string, $x^3/3$, $x^3/3$, and X^6/x.

Deciders for x^3/x has 9 possible configurations, or, in algebraic terms, permutations, and x^6/x has 36 possible configurations. This equates to R rational for x^3/x being $1/9$ and x^6/x being $1/36$.

This implies the existence of the map between Q rational to R rational.

3.9 Support Of Picking Function

The support of an inferrable language is now defined to be:

Dupport of PF of Q-rational-deciders = support(PF(Q-rational-deciders))

Decider in Q-rational-deciders such that decider is unique $\equiv$ Determinant of a configuration of a decider != 0

3.10 Law Of Strings

a = b + c	**b = a - c**	**-c = a - b**
$b = a - c$	$b - a = -c$	$-c + b = a$
$-c = a - b$	$b + c = a$	$-c - a = -b$

$(\boldsymbol{LHS} = RHS) \neq (\boldsymbol{RHS} \quad \mathbf{LHS})$
$(\boldsymbol{LHS} = \mathbf{RHS}) = (\boldsymbol{RHS} \quad LHS)$

Condense the above to get a generalization.

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

This is operating on variables, strings, and representations of it.

3.11 Commutativity Of Addition

$a + b = b - a \quad b - a = a + b$
$b + a = -a + b \quad -a + b = b + a$

$a + b \neq b - a \quad -a + b \neq b + a$
$b + a \neq -a + b \quad b - a \neq a + b$

Take the length function, length(s) $\equiv$ l(s), and apply to the '=' addition operations and see it's equivalence. Set b to a and reversal is accomplished.

$len(a) + len(b) = len(b) + len(-a) \equiv 11 = 11 \quad len(b) + len(-a) = len(a) + len(b) \equiv 11 = 11$
$len(b) + len(a) = len(-a) + len(b) \equiv 11 = 11 \quad len(-a) + len(b) = len(b) + len(a) \equiv 11 = 11$

$a + b \neq b - a \equiv 11 \neq 10 \qquad -a + b \neq b + a \equiv 01 \neq 11$
$b + a \neq -a + b \equiv 11 \neq 01 \qquad b - a \neq a + b \equiv 10 \neq 11$

3.12 Commutativity Of Multiplication

$a * b = b * -a \quad b * -a = a * b$
$b * a = -a * b \quad -a * b = b * a$

$a * b \neq b * -a \quad -a * b \neq b * a$
$b * a \neq -a * b \quad b * -a \neq a * b$

3.13 Additive Identity

$a = -a$ *for any a is equal to* $-a \qquad -a = a$ *where a and* $-a$ *is* 1
$a \neq -a$ *where a is* 1 *and* $-a$ *is* 0 $\quad -a \neq a$ *where a is* 1 *and* $-a$ *is* 0

The LHS and RHS are equations that test whether or not they are true or false, or in terms of computational complexity theory, it is satisfiable. 10 = 10 evaluates to true or 1. 01 != 10 evaluates to true or 1 too.

3.14 Multiplicative Identity

$a = -a$ where a and $-a$ are $1 \equiv$ a represented as monomial decider loops once
$a \neq -a$ where a is 1 and $-a$ is $0 \equiv$ a represented as a monomial decider loops once

3.15 Additive Inverse

The Additive Inverse

General Equivalence
$a + l = -a + r \equiv 10 = 10 \quad -a + r = a + l \equiv 10 = 10$
$l + a = r - a \equiv 10 = 10 \quad r - a = l + a \equiv 10 = 10$

Commutativity under Equivalence
$a + l = r - a \equiv 10 = 10 \quad -a + r = l + a \equiv 10 = 10$
$l + a = -a + r \equiv 10 = 10 \quad r - a = a + l \equiv 10 = 10$

General Reversal
$a + l \neq -a + r \equiv 10 = 01 \quad -a + r \neq a + l \equiv 01 = 10$
$l + a \neq r - a \equiv 10 = 01 \quad r - a \neq l + a \equiv 01 = 10$

Commutativity under reversal
$a + l \neq r - a \equiv 10 = 01 \quad -a + r \neq l + a \equiv 01 = 10$
$l + a \neq -a + r \equiv 10 = 01 \quad r - a \neq a + l \equiv 01 = 10$

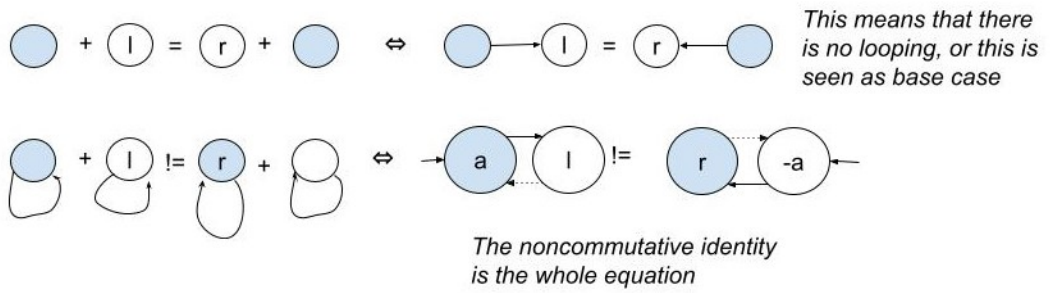

FIGURE 3.2: Additive inverse direction.

3.16 Multiplicative Inverse

Given a monomial such that it represents a monomial in a polynomial, if we loop around once, we see the identity path.

General Equivalence
$a * L = -a * R \equiv 10 = 10 \quad -a * R = a + L \equiv 10 = 10$
$L * a = R * -a \equiv 10 = 10 \quad R * -a = L * a \equiv 10 = 10$

Commutativity under Equivalence
$a * L = R * -a \equiv 10 = 10 \quad -a * R = L * a \equiv 10 = 10$
$L * a = -a * R \equiv 10 = 10 \quad R * -a = a * L \equiv 10 = 10$

General Reversal
$a * L \neq -a * R \equiv 10 = 01 \quad -a * R \neq a * L \equiv 01 = 10$
$L * a \neq R * -a \equiv 10 = 01 \quad R * -a \neq L * a \equiv 01 = 10$

Commutative under Reversal
$a * L \neq R * -a \equiv 10 = 01 \quad -a * R \neq L * a \equiv 01 = 10$
$L * a \neq -a * R \equiv 10 = 01 \quad R * -a \neq a * L \equiv 01 = 10$

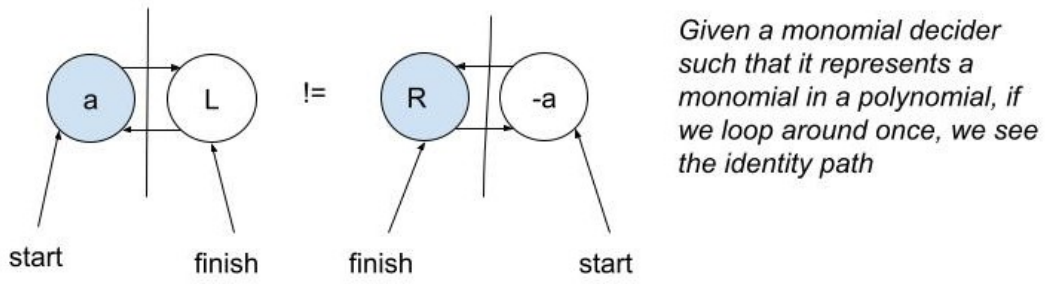

FIGURE 3.3: Multiplicative inverse direction.

3.17 Generalized Operations

The set of images that describes the operations on monomials can be put together to find a 2x2 matrix that describes them using the laws provided.

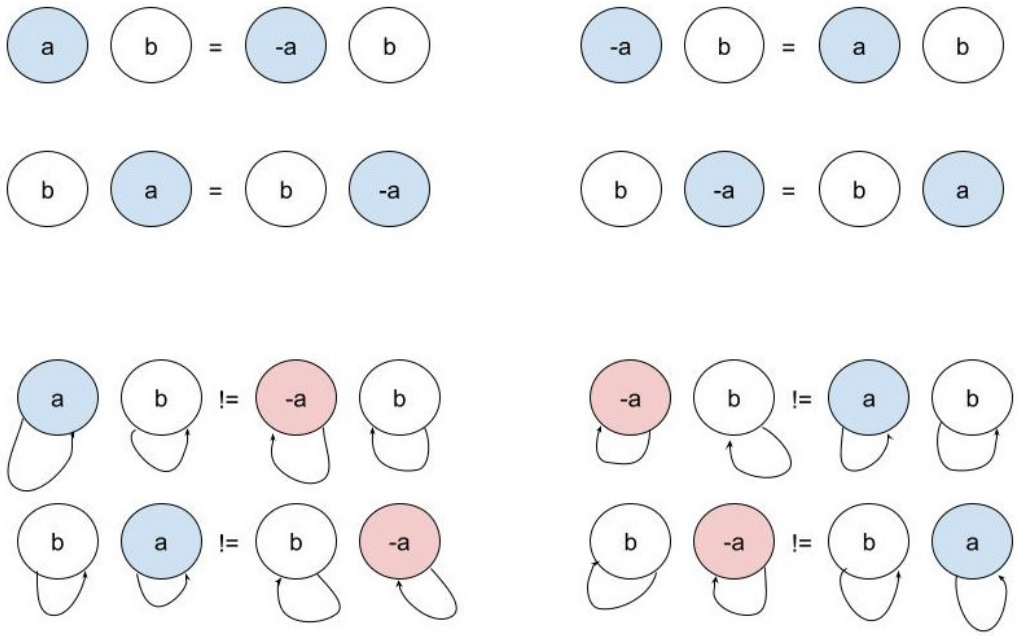

FIGURE 3.4: Generalized operations.

3.18 Generalized Communativity

For showing commutativity, have the following images to represent addition and multiplication. Fibonacci sequences are seen to be non-commutative if they are seen in this regards.

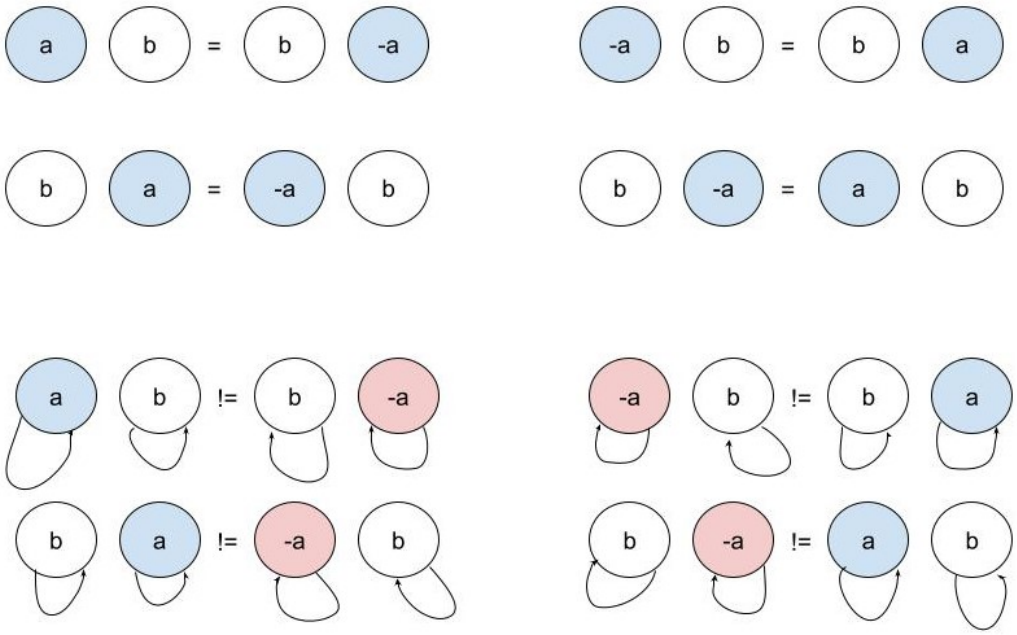

FIGURE 3.5: Communativity.

3.19 Associativity Of Addition

Associativity of addition is defined as:
a + (b + c) = (a + b) + c

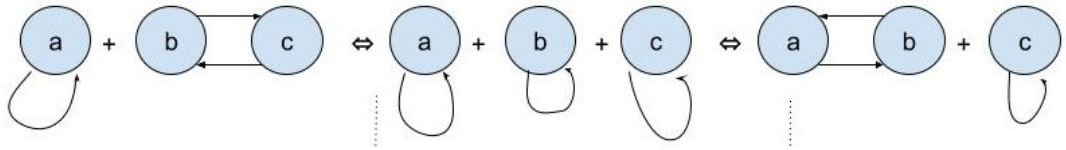

FIGURE 3.6: Associativity of addition.

a + bc

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

$b + c = -b + c \quad -b + c = b + c$
$c + b \neq -c + b \quad -c + b \neq c + b$

$c + b = c + b \quad -b + c = c + b$
$c + b \neq -c + b \quad -c + b \neq c + b$

$b + c = c - b \quad c - b = b + c$
$b + c \neq c - b \quad c - b \neq b + c$

$c + b = b + c \quad -c + b = b + c$
$c + b \neq b - c \quad -c + b \neq b + c$

a + b + c

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

$b = -b \quad -b = b$
$b \neq -b \quad -b \neq b$

$c = -c \quad -c = c$
$c \neq -c \quad -c \neq c$

ab + c

$$a+b=-a+b \quad -a+b=a+b$$
$$b+a\neq -b+a \quad -b+a\neq b+a$$

$$b+a=b+a \quad -a+b=b+a$$
$$b+a\neq -b+a \quad -b+a\neq b+a$$

$$a+b=b-a \quad b-a=a+b$$
$$a+b\neq b-a \quad b-a\neq a+b$$

$$b+a=a+b \quad -b+a=a+b$$
$$b+a\neq a-b \quad -b+a\neq a+b$$

$$c=-c \quad -c=c$$
$$c\neq -c \quad -c\neq c$$

3.20 Associativity Of Multiplication

Associativity of multiplication is defined as:
a * (b*c) = (a*b)*c

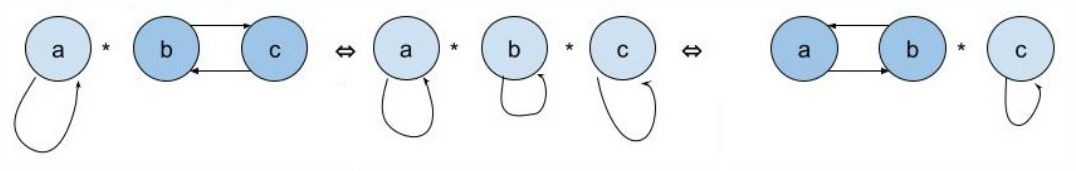

FIGURE 3.7: Associativity of multiplication.

a + bc

$$a=-a \quad -a=a$$
$$a\neq -a \quad -a\neq a$$

$$b*c=-b*c \quad -b*c=b*c$$
$$c*b\neq -c*b \quad -c*b\neq c*b$$

$$c*b=c*b \quad -b*c=c*b$$
$$c*b\neq -c*b \quad -c*b\neq c*b$$

$$b*c=c-b \quad c-b=b*c$$
$$b*c\neq c-b \quad c-b\neq b*c$$

$$c*b=b*c \quad -c*b=b*c$$
$$c*b\neq b-c \quad -c*b\neq b*c$$

a + b + c

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

$b = -b \quad -b = b$
$b \neq -b \quad -b \neq b$

$c = -c \quad -c = c$
$c \neq -c \quad -c \neq c$

ab + c

$a * b = -a * b \quad -a * b = a * b$
$b * a \neq -b * a \quad -b * a \neq b * a$

$b * a = b * a \quad -a * b = b * a$
$b * a \neq -b * a \quad -b * a \neq b * a$

$a * b = b - a \quad b - a = a * b$
$a * b \neq b - a \quad b - a \neq a * b$

$b * a = a * b \quad -b * a = a * b$
$b * a \neq a - b \quad -b * a \neq a * b$

$c = -c \quad -c = c$
$c \neq -c \quad -c \neq c$

3.21 Distibutivity

Distributivity is defined as:
a*(b+c) = a*b + a*c.

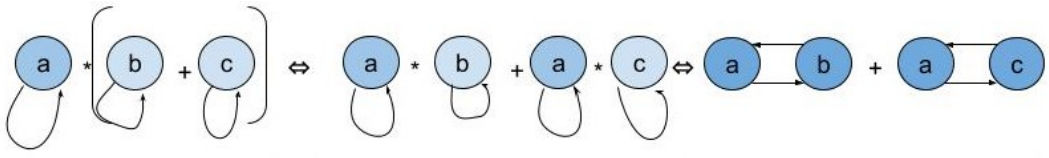

FIGURE 3.8: Associativity of multiplication.

a + b + c

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

$b = -b \quad -b = b$
$b \neq -b \quad -b \neq b$

$c = -c \quad -c = c$
$c \neq -c \quad -c \neq c$

a b + a c

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

$b = -b \quad -b = b$
$b \neq -b \quad -b \neq b$

$a = -a \quad -a = a$
$a \neq -a \quad -a \neq a$

$c = -c \quad -c = c$
$c \neq -c \quad -c \neq c$

a b + a c

$a * b = -a * b \quad -a * b = a * b$
$a * b \neq -a * b \quad -a * b \neq a * b$

$b * a = -b * a \quad -b * a = b * a$
$b * a \neq -b * a \quad -b * a \neq b * a$

$a * b = b - a \quad b - a = a * b$
$a * b \neq b - a \quad b - a \neq a * b$

$b * a = a * -b \quad -b * a = a * b$
$b * a \neq a * -b \quad -b * a \neq a * b$

$a * c = -a * c \quad -a * c = a * c$
$a * c \neq -a * c \quad -a * c \neq a * c$

$c * a = -c * a \quad -c * a = c * a$
$c * a \neq -c * a \quad -c * a \neq c * a$

$a * c = c - a \quad c - a = a * c$
$a * c \neq c - a \quad c - a \neq a * c$

$c * a = a * -c \quad -c * a = a * c$
$c * a \neq a * -c \quad -c * a \neq a * c$

3.22 Field

These rules result in a field defined by Wolfram.

Axioms	*Addition*	*Multiplication*
Commutativity	$a+b=b+a$	$a*b=b*a$
Identity	$a+0=a=0+a$	$a*1=a=1*a$
Inverses	$a+(-a)=0=(-a)+a$	$a*a^{-}1=1=a^{-}1*a\ if\ a!=0$
Associativity	$(a+b)+c=a+(b+c)$	$(a*b)*c=a*(b*c)$
Distributivity	$a*(b+c)=a*b+a*c$	$(a+b)*c=a*(b+c)$

This field is seen in *Garg, Gurvits, Oliveira, and Wigderson*.

Chapter 4

References

Garg, Ankit & Gurvits, Leonid & Oliveira, Rafael & Wigderson, Avi. (2016). A deterministic polynomial time algorithm for non-commutative rational identity testing. Annual Symposium on Foundations of Computer Science.

Weisstein, Eric W. "Field Axioms." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/FieldAxioms.html

Valiant, Leslie., Holographic Algorithms.

Berstel, J., & Reutenauer, C. (2010). Noncommutative rational series with applications. Cambridge University Press.

Sipser, M. (2006). Theory of Computation (2nd ed., p. 431). Course Technology, Cengage Learning.

Munkres, J. R. (2000). Topology (2nd ed., p. 537). Prentice Hall.

Artin, M. (2011). Algebra (2nd ed., p. 543). Pearson.

Enderton, H. B. (2001). Logic (2nd ed., p. 317). HARCOURT/ACADEMIC PRESS.

Lay, S. R. (2004). Analysis (4th ed., p. 394). Pearson Prentice Hall.

Cox, D. A., Little, J., & O'Shea, D. (1997). Ideals, varieties, and algorithms: An introduction to computational algebraic geometry and commutative algebra (2nd ed.). Springer.

Golub, G. H., & Van Loan, C. F. (1996). Matrix Computations (3rd ed., p. 728). Johns Hopkins University Press.

Hofstadter, D. R. (1999). Gödel, Escher, Bach: An eternal golden braid. Basic Books.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Shilov, G. E. (2012). Linear algebra (R. A. Silverman, Trans.). Dover Publications. (Original work published 1971)

Strang, G. (2009). Introduction to Linear Algebra (4th ed., p. 585). Wellesley-Cambridge Press.

Rosen, K. H. (2006). Discrete Mathematics And Its Applications (6th ed.). McGraw-Hill Education.

Prasolov, V. V. (1995). Intuitive topology. American Mathematical Society.

Carter, N. (2009). Visual group theory. Mathematical Association of America.

Spivak, M. (2008). Calculus (4th ed., p. 680). Publish or Perish.

Sullivan, M. (2008). Precalculus (8th ed., p. 1152). Prentice Hall.

Ung, E. (2023). A Language of Polynomials (Version 1.0.0).

Hamming, R. W. (1986). Numerical methods for scientists and engineers. Courier Corporation.

Weisstein, Eric W. "Field Axioms." From MathWorld–A Wolfram Web Resource.

Pinter, C. C. (2010). A book of abstract algebra (2nd ed.). Dover Publications.

Ung, E. (2023). Applications For Monomial Deciders (Version 1.0.1).

Ung, E. (2023). A Language Of Polynomials (Version 1.0.1).

Ung, E. (2018). Inferring Lindenmayer Systems (Version 2.0.1).