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 1, 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 re-framing 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.

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 $Decider < c \times x^n >\equiv c \times x^n = y$
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 $Decider < 5(2)^1 >\equiv 5(2)^0 = 5$

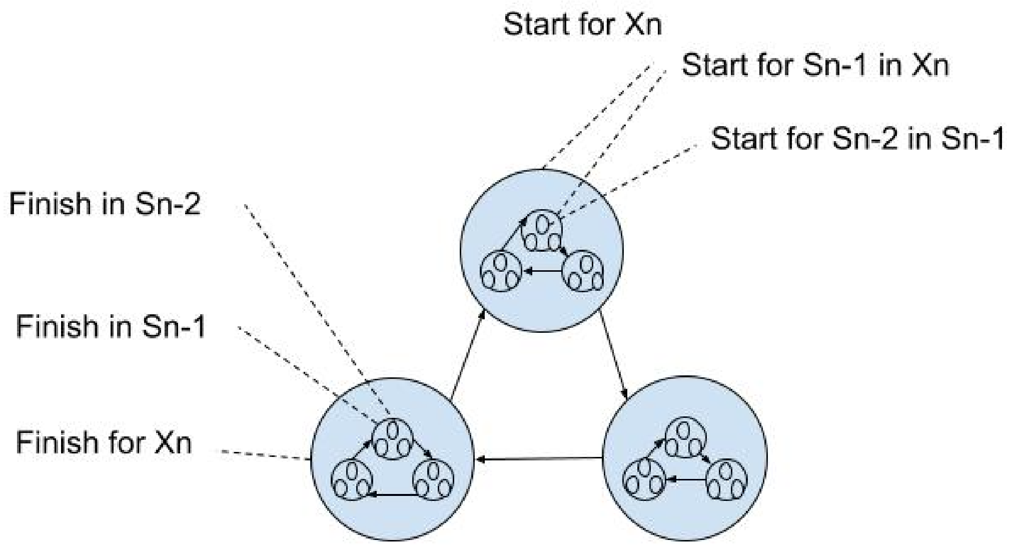

FIGURE 1.1: Decider that represents the monomial, x^3.

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.

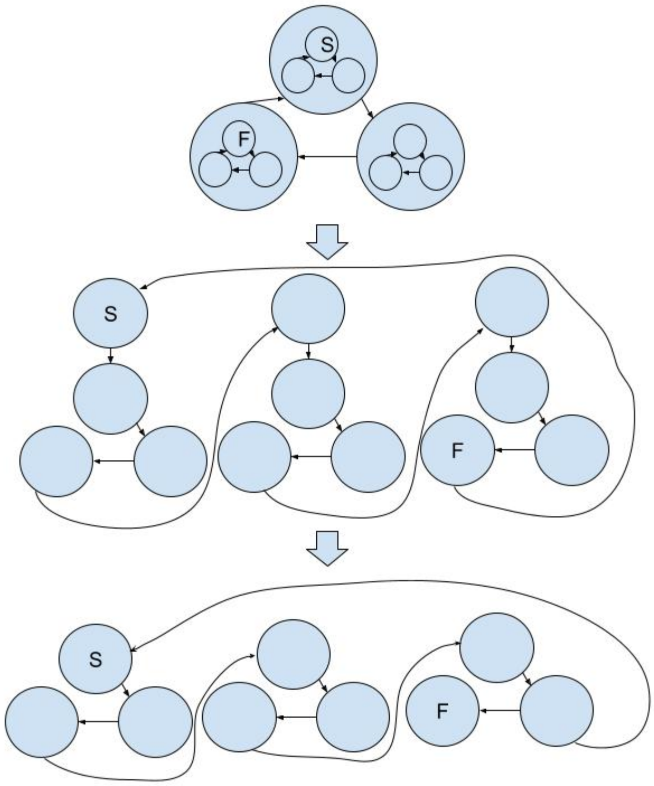

FIGURE 1.2: Top down removal for equivalence of decider and cyclic automata.

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.

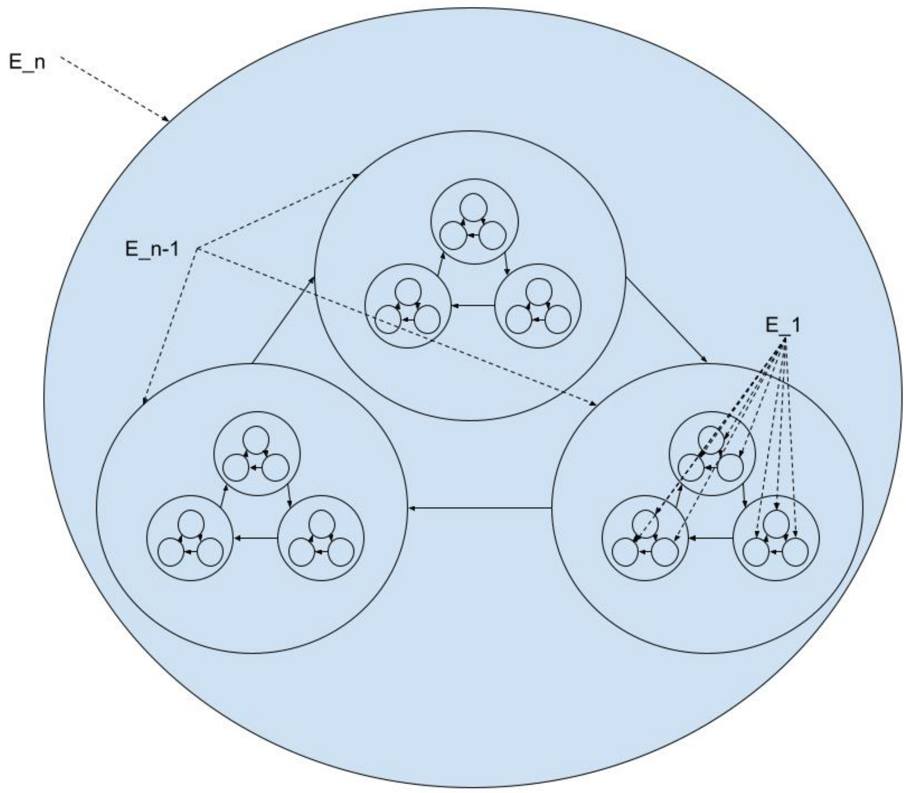

FIGURE 1.3: Visual example of what $\mathcal{E}_n$ of a rational expression.

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

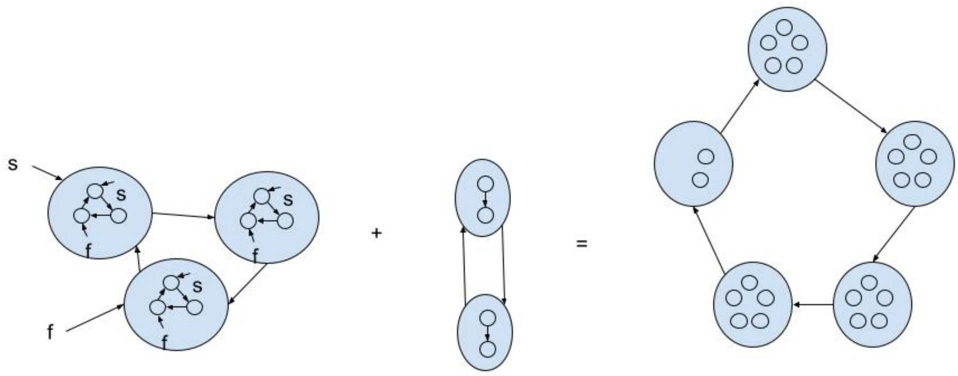

FIGURE 1.4: Addition of two deciders. The gradient of the circles remain the same after adding the two deciders together as the degree remains the same.

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

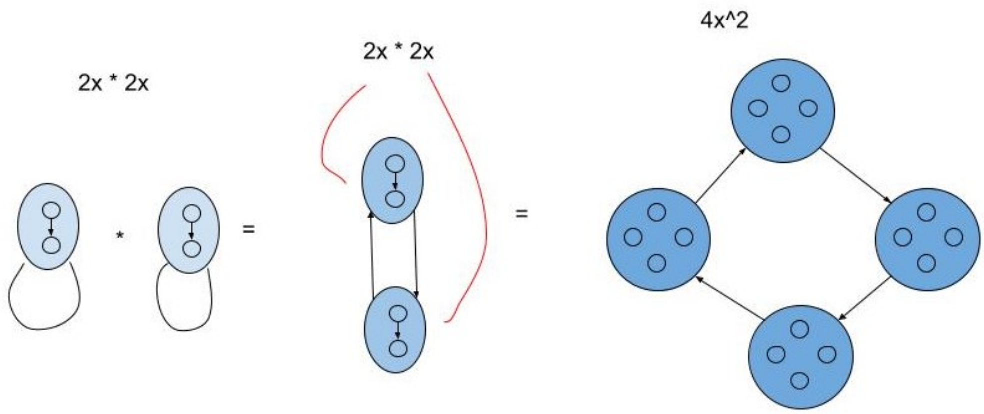

FIGURE 1.5: Product of two deciders. The gradient of the circles get denser after adding the two deciders together as the degree increases.

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.

$$[n \times n]\,[n \times n] = [m \times m] \text{ such that } m \neq n \text{ and } m, n \geq 0 \text{ and } m \leq n$$

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

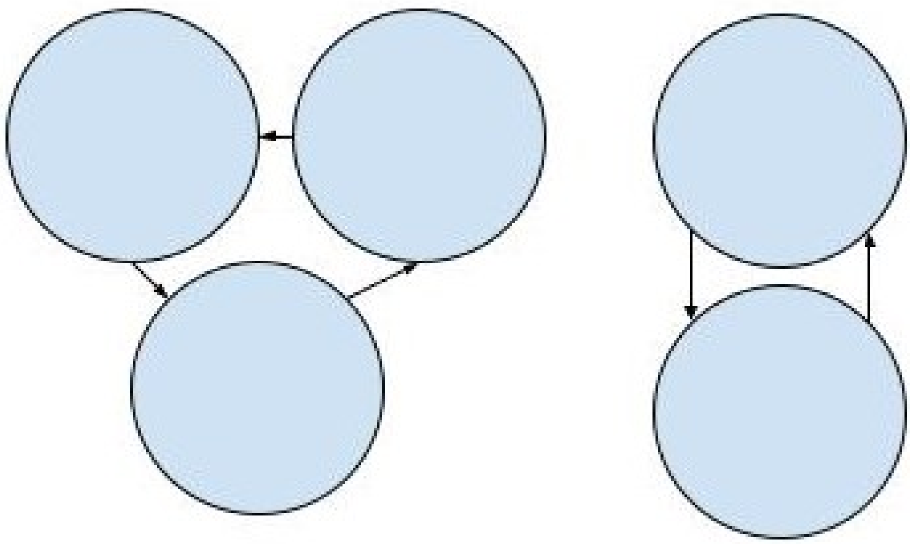

FIGURE 1.6: Multivariable monomial deciders can be seen treated as parallel processes running next to each other.

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

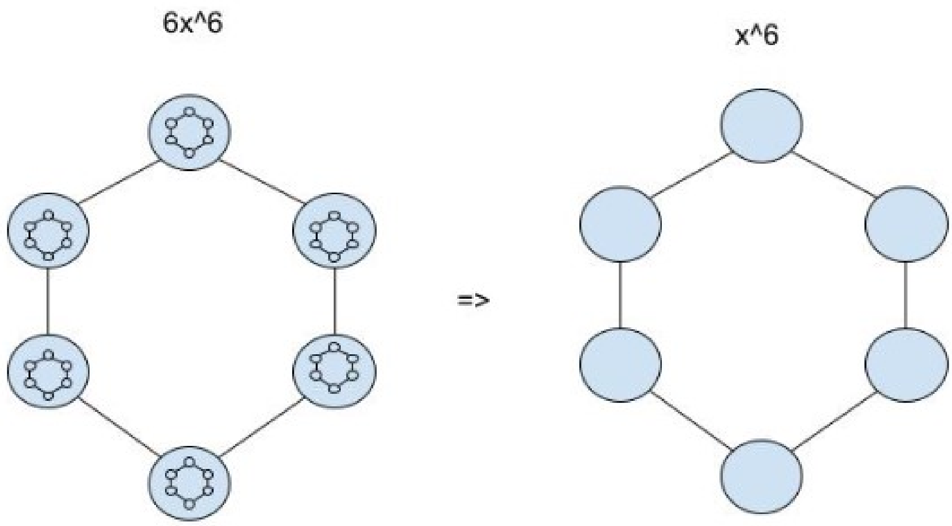

FIGURE 1.7: Generalization of a monomial decider.

A decider can be represented as the top layer only if short hand notation is necessary.

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

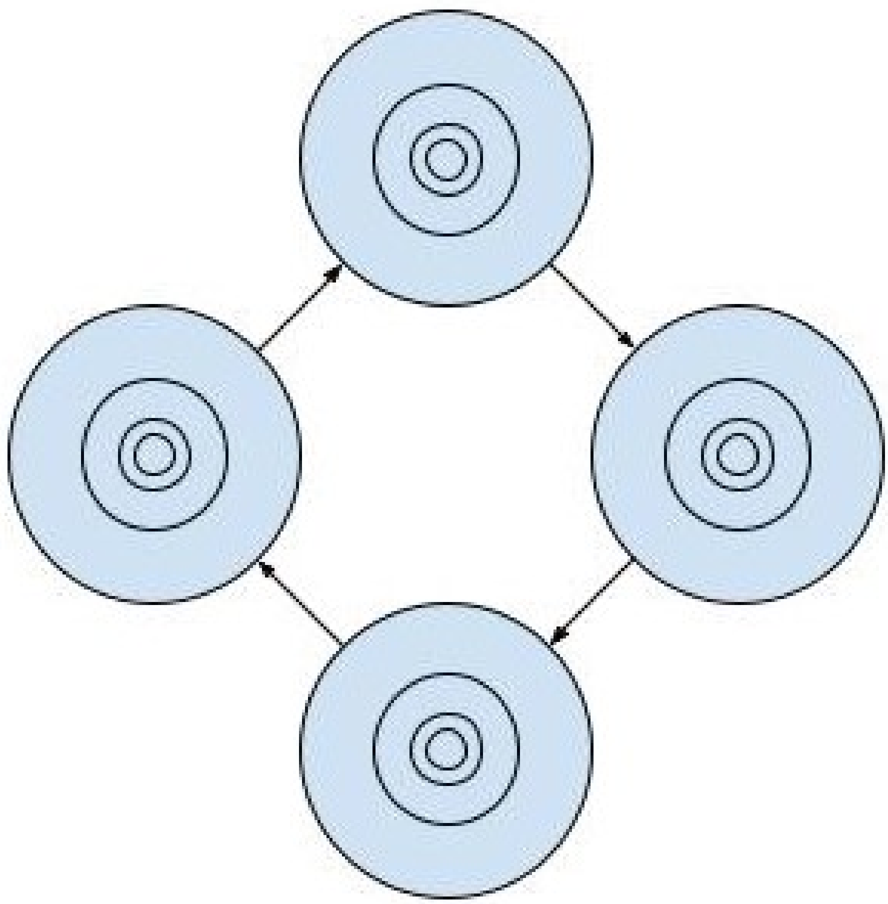

FIGURE 1.8: A concentric monomial decider is a generalized monomial decider with details missing.

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

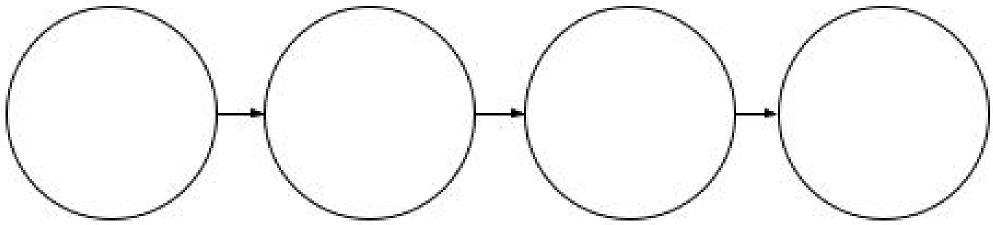

FIGURE 1.9: A constant represented as monomial decider, $Decider < cx^0 >$.

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.

1.11 Division

Division of monomial deciders

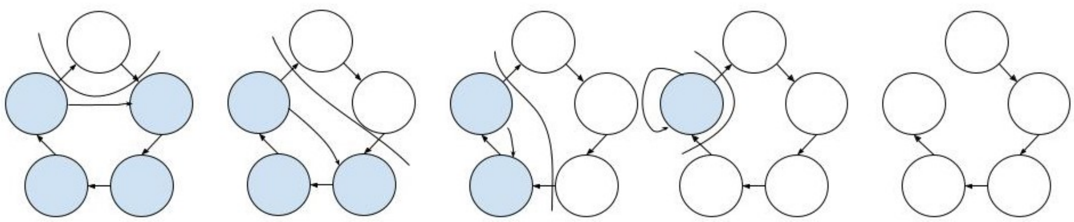

FIGURE 1.10: $Decider < x^5/x >$ to $Decider < x^5/x^5 >$.

$Decider < x^5/x > \equiv Decider < x^5 > /Decider < x^1 >$

$Decider < x^5/x^2 > \equiv Decider < x^5 > /Decider < x^2 >$

$Decider < x^5/x^3 > \equiv Decider < x^5 > /Decider < x^3 >$

$Decider < x^5/x^4 > \equiv Decider < x^5 > /Decider < x^4 >$

$Decider < x^5/x^5 > \equiv Decider < x^5 > /Decider < x^5 >$

In the following examples, $x_i \neq x_j$, meaning x_i is of a different representation than x_j

$Decider < x^5/x^1 > \equiv$ sequence of permutations of x_i, x_j such that the count of i is 4 and j is 1

$Decider(< x^5/x^2 > \equiv$ sequence of permutations of x_i, x_j such that the count of i is 3 and j is 2

$Decider < x^5/x^3 > \equiv$ sequence of permutations of x_i, x_j such that the count of i is 2 and j is 3

$Decider < x^5/x^4 > \equiv$ sequence of permutations of x_i, x_j such that the count of i is 1 and j is 4

$Decider < x^5/x^5 > \equiv$ sequence of permutations of x_i, x_j such that the count of i is 0 and j is 5

1.12 Multiple Divisions

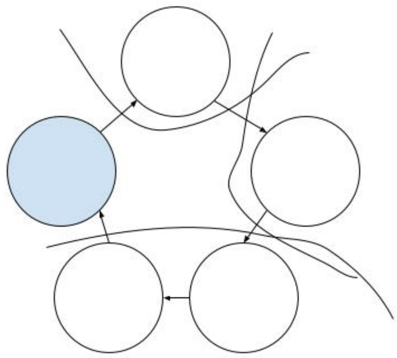

FIGURE 1.11: *Decider* $< x^5/x/x/x^2 >$

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

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

1.13 Equivalence

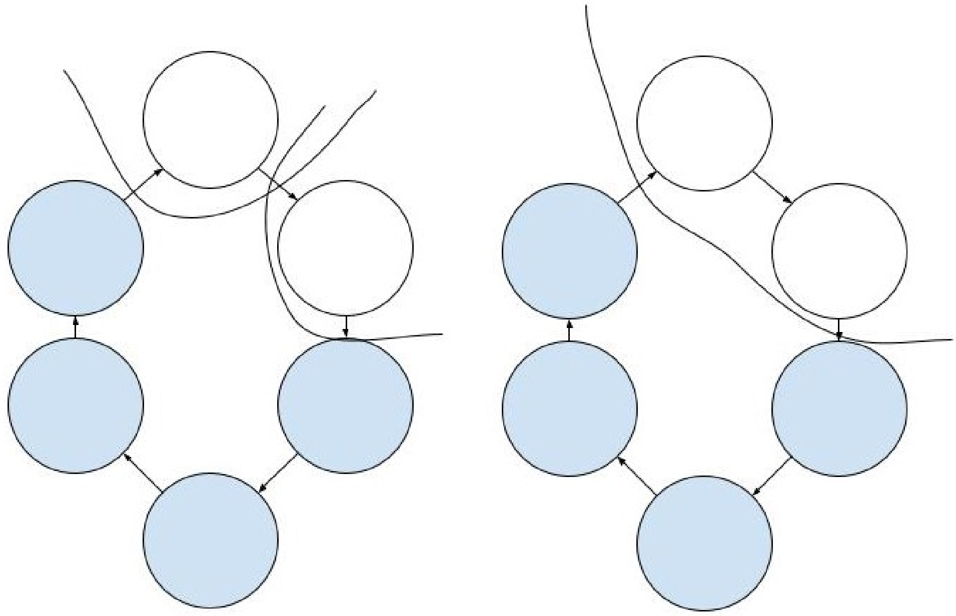

FIGURE 1.12: *Decider* $< x^6/x^2 >$

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.

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. Proof by construction visually to show a != b.

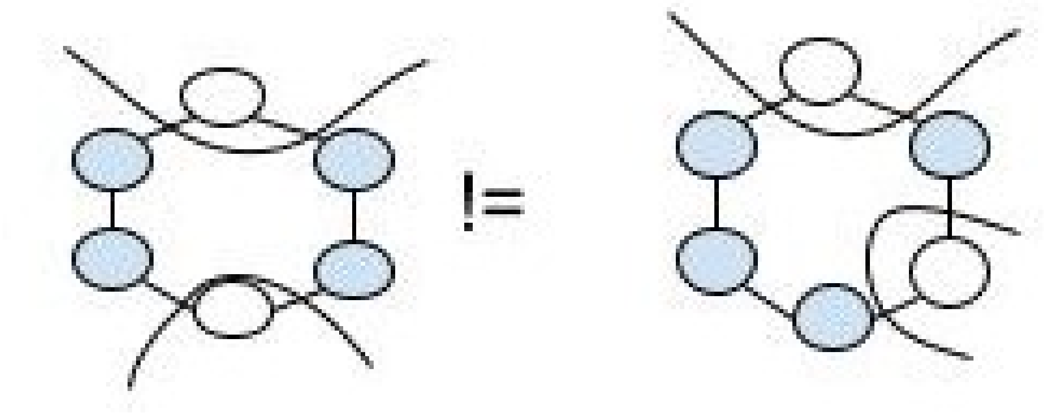

FIGURE 1.13: Two possible representations of *Decider* $< x^6/x/x >$.

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.

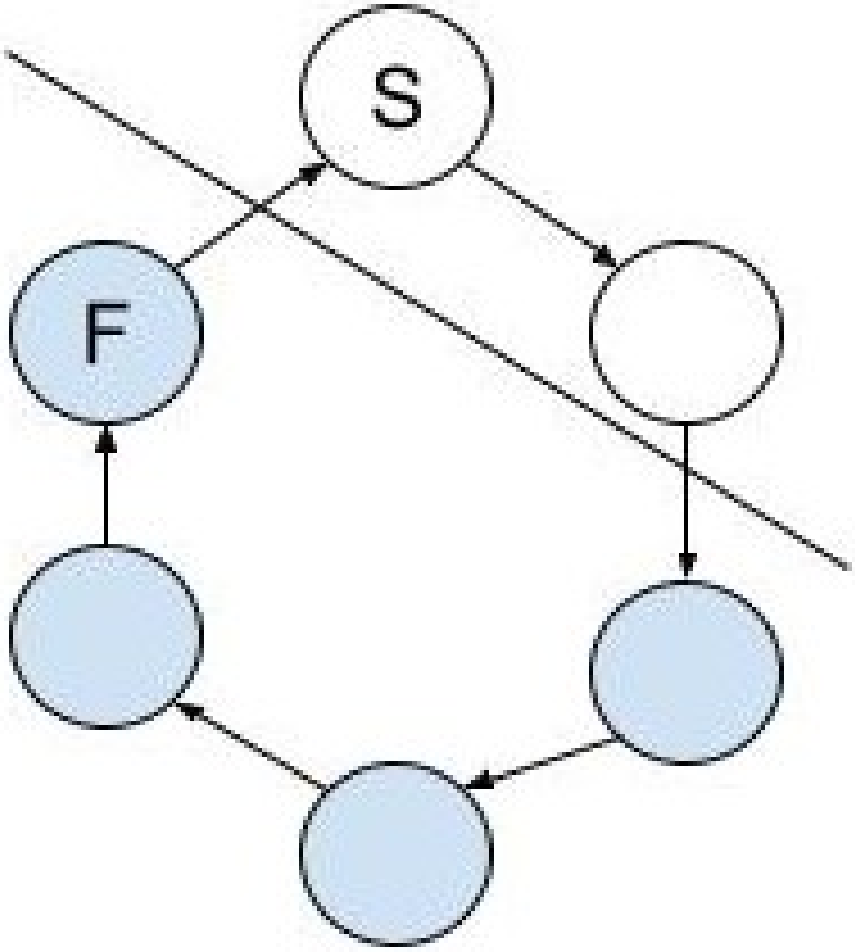

FIGURE 1.14: Path of one representation of *Decider* $< x^6/x^2 >$.

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

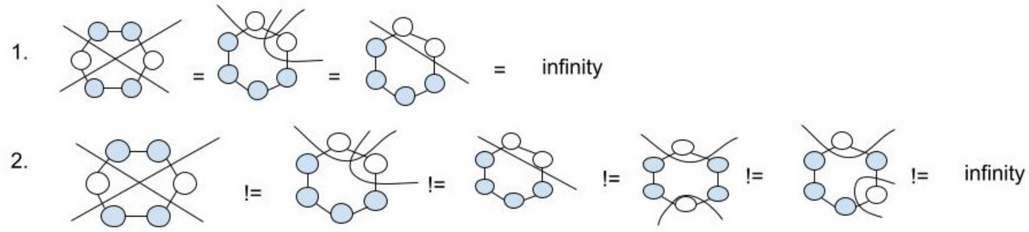

FIGURE 1.15: Godel's from *Decider* $< x^6/x^2 >$.

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

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 Starting With A Theorem Of Infiniteness

This section assumes that both P=NP and P!=NP and the following sections will provide reasoning and examples.

2.2 Mapping Out Representations

This section contains my opinions of what mathematics is.

2.3 Euler's Constant

Euler's constant is an example of P = NP because of it's use of calculus.

To show that it is also in the problem set of P $\neq$ NP, start by using the picking function going into infinity.

$$e = pf(x) = \theta(language\ of\ pf(x))$$

$$\{x = x^2/x = x^3/x^2 = x^4/x^3 = x^5/x^4 = \cdots = x^n/x^{n-1}\}$$

$$1 + 1 + 1/(1+1) + 1/(3+3) + 1/(4 + (4*3 + 4*2) + 4)$$

2.4 Sketching Into Code

There is a technique to develop code from a diagram of a decider. Take into account the degree of the states and that should account for the halting required to break from the loop.

```
bool generalizedMD(int y)
{
    if (y == 0)
    {
        return true;
    }
    var s = y;
```

```
    while (s >= 0)
    {
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                if (s == 0 && (j == 0 || j == 2) && i == 0)
                {
                    return true;
                }
                else if (s < 0)
                {
                    return false;
                }
                s -= 1;
            }
        }
    }
    return true;
}
```

2.5 Gather Some Data

Using the generalizedMD function where we make some sketch of the initial algorithm, we can collect some data Generate:
f(x) = y on the first line
Negatives on the second

We notice that we need two finishing states and that all the ven y's end in one state and all the odd y's end on the other.
E: Is it even?

We create a for loop of n^3 that with constant 2 and count how many times it passes through the finishing state.
T: Total amount of times it passes the finishing state.

Insight is gained by noticing that the difference only increases every other time and that it increases by a difference of 2 every it passes a finishing state.
D: Difference of the number of hits to the finishing state between the alst time it is called and the first.

```
// Generates negatives from a general monomial decider represented as an al
// that we can collect data about the negatives
int[] Generator(int max)
{
```

```
    int[] result = new int[max + 1];
    int x = 0;
    int negatives = 0;
    int i = 0;
    while (x < max + 1)
    {
        int num = 2 * (Convert.ToInt32(Math.Pow(x, 2)));
        if (generalizedMD(i))
        {
            // A simple verifier
            if (num == i)
            {
                Console.WriteLine(negatives);
                Console.WriteLine("f(" + x + ") = " + i);
                result[x] = i;

                x++;
                negatives = 0;
            }
            else
            {
                negatives++;
            }
        }
        i++;
    }

    return result;
}
```

2.6 Representing Monomial Deciders As Code

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

x is even at 0,8,32,72
x is odd at 2,18,50

There are four variables constructed:
Current hits records the number of times path traveled hits 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.

```
// After getting some log results, we can construct a decider
bool MonomialDecider2xx(int y)
{
    var total = 0;
    var hits = 0;
    var diff = 1;
    var isEven = 0;
    var s = 0;

    while (s <= y)
    {
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                for (int k = 0; k < 2; k++)
                {
                    if ((i == 0)&&(j == 0 || j == 1)&&(k == 0))
                    {
                        if (hits == total)
                        {
                            if (s == y)
                            {
                                Console.WriteLine(s + ": Hits: " + total);
                                return true;
                            }
                            else if (s > y)
                            {
                                return false;
                            }

                            total += diff;
                            isEven++;
                            if (isEven % 3 == 2)
                            {
                                isEven = 0;
                                diff += 2;
                            }
                        }

                        hits++;
                    }
                    s++;
                }
            }
        }
    }

    return false;
}
```

2.7 Negative Numbers

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

Cancellation

Addition

Start and Finish: $2x$

Start and Finish: x^2

Ignoring the rules for commutativity and association for now.

2.8 Pi

Representing the constant pi, π, in the language of polynomials using the Leibniz formula $\pi/4 = 1 - 1/2 + 1/5 - 1/7 + 1/9 + \cdots$

2.9 Fibonacci

Given the fibonacci sequence, $1, 1, 2, 3, 5, 8, 13, 21, \cdots$, representing this sequence as a monomial generator in the language of polynomials can be shown below.

From the circles 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.

Notice that on odd inputs, Ex, there aren't any circles in the circles/states but in even inputs on X, there are two states. On outputs, Ey, it is odd twice then even once.

```
int fibonacci(int n)
{
    if (n == 0)
    {
        return 0;
    }

    int y = 1;
    int y1 = 1;
    int y2 = 0;

    for(int i = 1; i < n; i++)
```

```
    {
        y = y1 + y2;
        y2 = y1;
        y1 = y;
    }

    return y;
}
```

2.10 Analysis Of Parity In Fibonacci

On analyzing the parity of Ex and Ey of the fibonacci sequence, we that there are two patterns.

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

$Det(E_1, E_2, E_3) = -1$
$Det(E_2, E_1, E_3) = 1$

$-1 + 1 - 1 = -1$
$1 - 1 + 1 = 1$

2.11 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 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.

```
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 + "\tsta
    }

    return stateX;
}
```

2.12 The Fibonacci Decider

Given the two monomial deciders we found using the determinant, we know that we must have six numbers in the sequence to decide if they form a Fibonacci sequence. In order to create this decider, we use an addition operator to merge them together because they cancel each other out. All deciders in each set must be true in order for the sequence to be a valid Fibonacci sequence.

2.13 The Fibonacci Picking Function

Now, let's apply the picking function, pf, to the fibonacci decider and show the probability, P_r, of finding a sequence. $pf(xyz_1 + xyz_2) = Pr(xyz_1) * Pr(xyz_2) \leq 1/n^k$

Each decider has 3 representations giving $3^2 = 9$ total for each set. We pick one decider in each set to get the probability.

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

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.

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

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

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.

There are two deciders on the right hand side.

3.4 Fibonacci DOL Decider Right Hand Side

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.

3.5 The Law of Commutativity and Noncommutativity

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.

3.6 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

3.7 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.

3.8 Support Of Picking Function

The support of an inferrable language is now defined to be:
support of PF of Q-rational-deciders = support(PF(Q-rational-deciders))
decider in Q-rational-deciders such that decider is unique $\equiv$ det(decider) != 0
The decider chosen has an equation that is bold and italicized and there is a set of equations that are distinctly bold and italic that complete this equation to form the decider.

3.9 Law Of Strings

Although the law of commutativity and noncommutativity is a theorem, it's helpful to get a big picture view. Let's condense the law of commutativity and noncommutativity down even further to find some laws.

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

3.10 Commutativity Of Addition

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.

3.11 Commutativity Of Multiplication

Commutativity of Multiplication

3.12 Additive Identity

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.

3.13 Multiplicative Identity

The Identity of Itself on Multiplication

3.14 Additive Inverse

The Additive Inverse
General Equivalence
Commutativity under Equivalence
General Reversal
Commutativity under reversal

3.15 Multiplicative Inverse

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

3.16 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.

3.17 Generalized Communativity

For showing commutativity, have the following images to represent addition and multiplication.

3.18 Associativity Of Addition

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

3.19 Associativity Of Multiplication

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

3.20 Distibutivity

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

3.21 Field

The Field Axioms are defined as:

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:
`\hypersetup{urlcolor=red}`, or
`\hypersetup{citecolor=green}`, or
`\hypersetup{allcolor=blue}`.
If you want to completely hide the links, you can use:
`\hypersetup{allcolors=.}`, or even better:
`\hypersetup{hidelinks}`.
If you want to have obvious links in the PDF but not the printed text, use:
`\hypersetup{colorlinks=false}`.