

Shawnee State University

Digital Commons @ Shawnee State University

Master of Science in Mathematics

College of Arts & Sciences

Summer 2019

A Conceptual Framework for Equation Solving Strategies

Julius Alex Kosan

Follow this and additional works at: https://digitalcommons.shawnee.edu/math_etd



Part of the [Mathematics Commons](#)

A Conceptual Framework for Equation Solving Strategies

By J. Alex Kosan

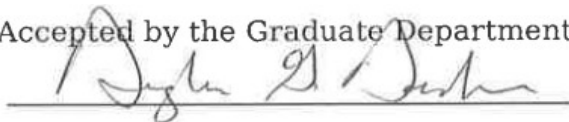
Department of Mathematical Sciences

Submitted in partial fulfillment of the
requirements for the degree of

Master of Science, Mathematical Sciences

Date: 08/05/19

Accepted by the Graduate Department

A handwritten signature in black ink, appearing to read "Dustin G. Decker", is written over a horizontal line.

Graduate Director, Date

The thesis entitled 'A Conceptual Framework for Equation Solving Strategies' presented by Julius A. Kosan, a candidate for the degree of Master of Science in Mathematical Sciences, has been approved and is worthy of acceptance.

8/6/2019

Date

Dr. J. S. Sel

Graduate Director

8/5/19

Date

Julius A. Kosan

Student

Table of Contents

1	Introduction.....	7
1.1	Background of the Problem.....	8
1.2	Statement of the Problem.....	10
1.3	Purpose of this Paper.....	11
1.4	Significance of this Paper.....	11
1.5	Primary Research Questions.....	12
1.6	Overview of Conceptual Framework.....	12
1.7	Overview of Measurement Programs.....	14
1.8	Assumptions.....	16
1.9	Limitations.....	17
1.10	Definition of Terms.....	17
1.11	Summary.....	18
2	Literature Review.....	19
2.1	Meta-Level Inference for PPEs.....	19
2.1.1	History of PRESS.....	20
2.1.2	The Implementation of PRESS.....	22
2.1.2.1	Goal of Equation Solving.....	23
2.1.2.2	Meta-Level Strategies of PRESS.....	24
2.1.2.2.1	Isolation.....	24
2.1.2.2.2	Collection.....	27
2.1.2.2.3	Attraction.....	28
2.1.2.2.4	The Basic Method.....	29
2.1.3	History of Other PPEs.....	29
2.1.3.1	MathXpert.....	30
2.1.3.2	MathSteps.....	32
2.2	Comparing CASs.....	32
2.3	Intelligent Tutoring Systems (ITS).....	34
3	The Theoretical Framework.....	37
3.1	Preliminary Definitions and Facts.....	37
3.1.1	Definition of a Solved Equation.....	37
3.1.2	Definition of an Equation Step-by-Step Solution.....	38
3.2	PPEs Algorithmic Facts and Attributes.....	39
3.2.1	An Outline of a Model for PPEs.....	39
3.2.1.1	Algorithms.....	39
3.2.1.2	Meta-Level Control.....	40
3.2.1.3	Categorizing Rewrite Rules.....	41
3.2.1.4	Sameness Through Transformation.....	42
3.2.2	The Necessity of Depth Reducing, Collection and Pre-Collection Rules	42

3.2.2.1	Depth Reducing Rules.....	43
3.2.2.2	Collection Rules.....	45
3.2.2.3	Pre-Collection Rules.....	45
3.3	PPES Output Facts and Meta-Level Functions and Predicates.....	46
3.3.1	Sub-Equation Meta-Level Functions and Predicates.....	47
3.3.2	Single-Step Meta-Level Functions and Predicates.....	48
3.3.2.1	Instances of the Unknown Meta-Level Function.....	48
3.3.2.1.1	Positions of Unknowns Function.....	48
3.3.2.1.2	Number of Unknowns Function.....	48
3.3.2.1.3	Smallest Unknown-Containing Subtree Function.....	49
3.3.2.1.4	Maximum Depth of Unknown Function.....	49
3.3.2.1.5	Minimum Depth of Unknown Function.....	49
3.3.2.1.6	Distance Between All Unknowns Function.....	49
3.3.3	Multi-Step Meta-Level Functions and Predicates.....	50
3.3.3.1	Consecutive Step Functions.....	50
3.3.3.1.1	Number of Unknowns Change Function.....	50
3.3.3.1.2	Maximum Depth Change Function.....	50
3.3.3.1.3	Distance Change Function.....	51
3.3.4	Single-Solution Meta-Level Functions.....	51
3.3.4.1.1	Number of Steps Function.....	51
3.3.5	Multi-Solution Meta-Level Functions and Algorithms.....	51
3.3.5.1	Random Equation and Statistics Approach.....	52
3.3.5.2	Pattern Recognition Approach (Not Implemented).....	52
3.3.5.3	Intelligent Modeling Approach (Not Implemented).....	52
4	Implementation of Measurement Programs in MathPiper.....	54
4.1	Construction of Experiments.....	55
4.2	Functionality of MathPiper.....	55
4.3	Generating Random Equations.....	57
4.3.1	Naive Method.....	58
4.3.2	Shortcomings of the Naive Method.....	61
4.3.2.1	Attributes Not Normally Distributed.....	62
4.3.2.2	Undesirable Correlations of Numerical Attributes.....	65
4.3.3	Ways to Improve the Naive Method.....	65
4.4	Regression Models.....	67
5	Conclusion.....	70
5.1	Summary.....	70
5.1.1	Construction of the Conceptual Framework.....	70
5.1.2	Implementing Measurement Programs.....	72
5.2	Shortcomings.....	74
5.3	Future Research.....	76
6	Appendices.....	79
6.1	Appendix A: Meta-Level Languages and Object-Level Languages.....	79

6.1.1	Formal Languages.....	79
6.1.2	Languages to Talk About Languages.....	80
6.1.3	Meta-Languages.....	81
6.1.3.1	Object-Level Domain of Discourse.....	81
6.1.3.2	Object-Level Sentences.....	82
6.1.3.3	Meta-Level <i>Domain of Discourse</i>	82
6.1.3.4	Meta-Level Sentences.....	82
6.2	Appendix B: The Language of Expression Trees.....	84
6.3	Appendix C: Pattern Matching and Rewrite Rules.....	94
7	Bibliography.....	96
8	Biography.....	98

Illustration Index

Figure 1 Abstract Expression Tree Database.....	16
Figure 2: Step by Step Solution.....	24
Figure 3: Line 3.....	25
Figure 4: Line 4.....	25
Figure 5: Line 5.....	26
Figure 6: Line 6.....	26
Figure 7: Line 3.....	27
Figure 8: Line 2.....	27
Figure 9: Line 1.....	28
Figure 10: Abstract Expression Tree Databases.....	58
Figure 11: Starting Maximum Depth Histogram.....	62
Figure 12: Starting Total Distance Histogram.....	63
Figure 13: Step Count Histogram.....	63
Figure 14: Depth and Distance Regression Line.....	65
Figure 15: Depth and Steps Regression Line.....	68
Figure 16: Distance and Steps Regression Line.....	68
Figure 17: Expression Tree.....	85
Figure 18: Nodes.....	86
Figure 19: Arcs.....	87
Figure 20: Root Node.....	87
Figure 21: Child Nodes.....	88
Figure 22: Leaf Nodes.....	89
Figure 23: Node Positions.....	90
Figure 24: Dominant Node.....	91
Figure 25: Node Position.....	92
Figure 26: Node Position.....	92
Figure 27: Path Length.....	93
Figure 28: Depths.....	94

1 Introduction

The subject of elementary algebra, hereafter referred to "algebra", is considered a fundamental skill in the civilized world. Equation solving in particular is one of the most basic skills that is taught in the subject of algebra. While the axioms and inference rules of algebra are well established and agreed upon, the methods of how to employ these axioms and rules are not taught using an explicit system [Bundy 1975]. It is known that when a person attempts to solve an equation they must be, to some degree, using some implicit system to select each new step. It is evident that they are not simply applying the axioms, theorems and inference rules of algebra randomly and indiscriminately, otherwise their "solutions" would be useless. Two questions then arise: what is the nature of these implicit systems, and how can more be learned about them?

While it is clear that one or more systems for solving equations are in use by humans, and by computers that generate human like solutions, these systems are usually opaque or difficult to understand. Humans cannot explain in any great detail how they are able to select the correct steps while solving equations [Bundy 1975]. Software that is designed to mimic human equation solving is usually not inspectable because it is closed source. While research into the nature of systems that can be used both by humans and computers exists, it did not develop the concepts necessary to apply it to education research. [Bundy A 1983]. Therefore, this paper describes an initial conceptual framework for describing different aspects of these systems that can be used by educational

researchers.

1.1 Background of the Problem

This paper will refer to systems for solving equations that can be understood and followed by humans and be implemented by computers as psychologically plausible equation solvers (PPES). Research into characterizing and designing PPES systems started in the early 1970s. Dr. Alan Bundy was the first to consider the problem of creating an equation solver that mimicked typical human equation solving methods [Bundy 1975]. He was studying meta-level inference for automatic theorem provers in the burgeoning field of artificial intelligence at The University of Edinburgh. The problem that early theorem provers presented to researchers at that time was they were not capable of selecting the correct steps to take in their inferences. A standard method that is used in classical artificial intelligence to navigate the exponential explosion of possible steps is heuristics or “rules of thumb” [Nilsson 1980]. However, Bundy wished to take a more scientific approach to the problem, and he selected the technique of meta-level inference.

Bundy believed that a system for mimicking human equation solving would be a appropriate entry point into meta-level inference research because equation solving was considered to be well understood. However, when he consulted the literature, he found that no explicit system existed for solving equations [Bundy 1975]. While the axioms and transformation rules were established, there were no explicit methods recorded for how humans should solve equations. Further, when Bundy interviewed mathematicians, they could not explain how they were

able to quickly and directly solve equations. This led Bundy and other researchers at Edinburgh to create the equation solving system PRESS (PRolog Equation Solving System) which used meta-level inference. PRESS uses a meta-level description of the goal of solving an equation in order to select the steps in a solution, and it uses meta-level strategies to achieve this meta-level goal. The researchers at Edinburgh advanced PRESS to the point that it could solve most high school level algebra problems [Sterling L, Bundy A, Byrd L, O'Keefe R and Silver B 1982].

While it was the desire of the researchers at Edinburgh for the ideas implemented in PRESS to be used in education [Bundy personal correspondence, Saturday, July 2, 2016], this has not occurred. One reason for this is that the original researchers were primarily interested in experimenting with meta-level inference, and working to apply PRESS to education was outside of their research goals. PRESS's association with classical artificial intelligence has also likely prevented it from becoming well known. PRESS was created during the initial surge in artificial intelligence research. When it was found that artificial intelligence was considerably more difficult to create than was previously thought, a shortage of funding resulted known as the "AI winter". Many projects lost funding, and artificial intelligence programs, like PRESS, were considered failures. This led most educational institutions to stop teaching classical artificial intelligence topics. Thus, very few people today are capable of understanding and developing systems like PRESS. Other problems include: the general lack of software platforms capable of implementing PRESS's ideas, and a resistance to

read the relevant literature. For these reasons, the research that PRESS is based on has largely been forgotten.

While PRESS was the earliest PPES, and the one most heavily based on published research, there are several other systems that have been created more recently. Examples include the following:

System	Open Source	Based on PRESS	Reference
MathXpert	No	Yes	helpwithmath.com
MathSteps	Yes	No	github.com/socraticorg/mathsteps
MathPapa	No	?	mathpapa.com
Wolfram Alpha	No	?	wolframalpha.com
Symbolab	No	?	symbolab.com
Presston	Yes	Yes	mathpiper.org

Table 1: List of PPESs

As this table indicates, most of these systems are not open source. This means their capabilities cannot be measured by inspecting their source code. Therefore, any useful measurement of these systems must be based on their inputs and outputs. However, no conceptual framework or measurement system has been created yet that can be used to understand these PPESs.

1.2 Statement of the Problem

While the conceptual framework that PRESS is based on provides many insights into the nature of PPESs, it is still limited. A well developed conceptual framework for describing the full range of possible PPESs, including those in the above list, does not currently exist. This prevents focused deliberate

improvement or comparison of PPES systems for use in education. While the Edinburgh conceptualization of PPESs is advanced, it is specialized for the creation of PRESS. A more general conceptualization that accounts for more of the possible variations found in different PPESs, both human and automated, is required to advance educational research related to equation solving.

1.3 Purpose of this Paper

This paper describes and partially justifies the implementation of an initial conceptual framework for human and computer PPESs. Without such a conceptual framework, it is difficult to describe the characteristics of different PPES systems. This paper seeks to identify concepts and facts that are applicable to PPESs. This paper will also consider the software-based measurement of these concepts by using meta-level descriptions of the relationship between the input equations and their output solutions. Not every attribute of PPESs that will be considered will have an associated measurement program, but some attempt will be made to describe how such a measurement might be implemented in software. The aim of this initial conceptual framework and the related measurement programs is not completeness but instead is to provide a starting point for further research using more sophisticated methods.

1.4 Significance of this Paper

The conceptual framework that is used to characterize PPESs, and the resulting measurement programs, proposed in this paper are meant to help education researchers create theories about PPESs. One application of the

proposed framework is to provide a starting point for building systems that can show the strengths and weaknesses of automatic PPEs. These results could then be used to select or improve them. Another application is helping students improve their understanding of equation solving. Measurements of students based on this conceptual framework could be used to construct a model of the student's understanding. This model could then be used to advance the student by providing useful explanations or assigning appropriate problems. If this framework is developed to a sufficient degree, it would provide the language necessary to improve the teaching of equation solving in general.

1.5 Primary Research Questions

- What are some concepts and facts that are useful for characterizing PPEs?
- How can these concepts be measured by software?

1.6 Overview of Conceptual Framework

This paper primarily seeks to identify concepts related to the characterization of PPEs that can be incorporated into the theories of future research. The primary concepts developed in this paper will be the attributes of the equations and their solutions which are produced by PPEs. This paper does not offer a comprehensive theory for the usefulness of PPEs. It instead aims only to identify concepts and facts from both the literature and from observation that can form a language which can be used in future theories about PPEs.

The concepts in the conceptual framework pull heavily from Tarski's theory of

stratified metalevels [Sowa 2000 p.320] as well as Bundy's research into meta-level inference for PPESs. The concepts in this conceptual framework have been selected for their potential usefulness in creating educational material, measuring automatic PPESs and building intelligent tutoring systems (ITS).

This framework seeks to describe the attributes inherent in PPESs.

However, most of the attributes that would be of interest cannot be measured directly from the PPES. This is because most PPESs are closed source, in the case of computers, or subconscious, in the case of humans. Therefore structures in solutions will be included as concepts along with select attributes of these structures. The conceptual framework will seek to describe these attributes by categorizing them into different "tiers" of structures built of smaller structures. Attributes obtained from multiple solutions to equations from a given PPES will therefore be used to infer attributes of the PPES in the measurement programs. Each solution will contain steps. Each step will, usually, be an equation with one unknown. Each equation will contain sub-trees (see Appendix B). The conceptual framework measures attributes of the following structures:

- Subtree attributes
- Single step attributes
- Multiple step attributes
- Single solution attributes
- Multiple solution attribute
- Whole system attributes

The evaluation of attributes at each tier will be used in the attributes from sub-tiers (i.e. multiple step attributes will be defined in terms of single step attributes). The conceptual framework will then focus on defining potentially useful attributes of the structures at each level (it should be noted that a PPES does not contain any of these structures, it generally generates them when given an equation and is therefore more like an intension than an extension).

1.7 Overview of Measurement Programs

For those concepts that will have a measurement program described, the measurement program will be written using the MathPiper computer algebra system (CAS). There are several reasons for this. MathPiper is an education-oriented CAS that has meta-language support. Without meta-language support, no description of PPESs would be possible. MathPiper facilitates experimentation due to its worksheet format, and it is also open source which makes the results from this paper easier to access by future researchers. In addition, MathPiper has a PRESS inspired PPES called Presston which allows easy testing of the measurement programs created in this paper. MathPiper is therefore well-suited for this type of research.

Presston is still in development, and does not implement the whole of PRESS. However, Presston makes several additions to the methods used in PRESS in order to increase the detail of the solutions it produces. Presston uses a combinations of rewrite rules and symbolic procedures in order to solve equations. Since Presston is written in MathPiper, it avoids a problem that is

inherent with all computer algebras systems that are not designed for use in education, which is their automatic evaluation of expressions. Being written in MathPiper also facilitates experimentation by enabling the solver and the test code to easily interact. Generation of random equations is performed with procedures that are already included with MathPiper, and it also enables Preston to output LaTeX and other formatted information about the solutions it produces, which is useful for debugging.

The main process that has been selected for testing these measurement programs is based on statistics. There are three steps in this process:

1. Generate a random equation.
2. Solve this equation using Preston and record the step-by-step solution.
3. Measure certain attributes of both the equation and the solution.

The data that will be used for the testing of the measurement programs will be generated using pseudo random processes. The data produced will be equations that contain addition, subtraction and multiplication. Note that division has not been used because Preston does not yet contain the rules that are necessary to solve equations containing division. Equations are generated using an abstract expression tree database that is included in MathPiper. Abstract expression trees, in this instance, refer to a tree structure whose root node is an equals sign, has operator placeholders for the non-leaf nodes and number placeholders for the leaves. The abstract expression tree database is exhaustively generated to a certain depth (see Figure 1).

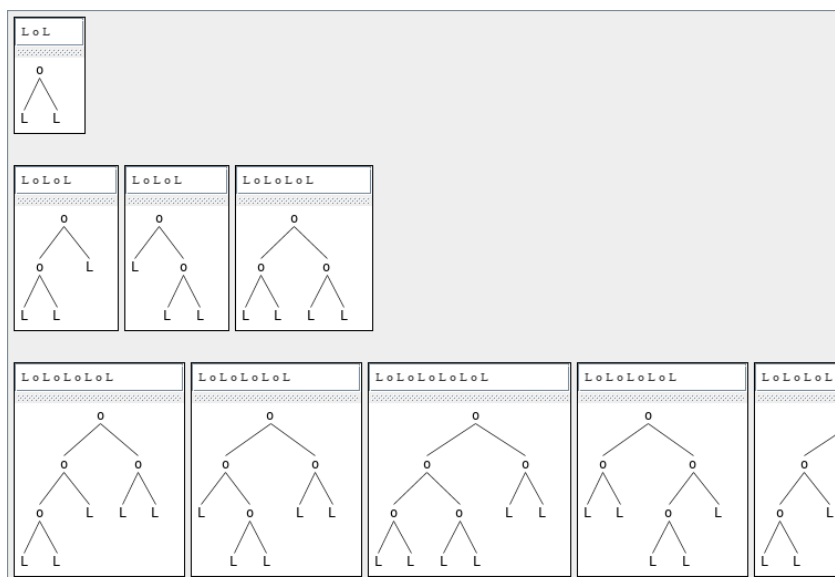


Figure 1 Abstract Expression Tree Database

Each equation is generated by randomly selecting an abstract expression tree from the database and then randomly replacing each of the nodes with different operators and randomly replacing each of the leaves with constants and instances of the unknown. <Insert more example images> The number of instances of the unknown can be controlled. For example, if the researcher wished to create equations with exactly three unknowns, this can be done. These equations are then solved using Presston. The result of these solutions is a list of the steps taken. After each equation is solved, a collection of attributes is measured from both the structure of the equation and the structure of the solution.

1.8 Assumptions

There are several assumptions that the proceeding papers makes. It is assumed that there are no inconsistencies with the MathPiper language,

Presston or the random equation generator.

1.9 Limitations

There are two primary limitations of this paper. First, the conceptual framework is incomplete. This is to expected from an initial consideration of this topic. Second, sophisticated methods for creating measurement programs, such as pattern recognition, optimization, and using theorem provers to theorize about missing steps, have not been used. Other limitations includes the relatively simple form of the equations solved due to the incompleteness of Presston.

1.10 Definition of Terms

Note that many of the definitions for terms in this paper are covered in the appendices.

- **Appendix A: Meta-Level Languages and Object-Level Languages**
 - Meta-level
 - Object-level
 - Meta-strategy
 - Meta-description
- **Appendix B: The Language of Expression Trees**
 - Depth
 - Distance
 - Abstract expression tree
- **Psychologically Plausible Equation Solver (PPES):** Any algorithm that gives a step-by-step solution to an equation in a manner that either a human can understand and follow or at least mimics typical human equation solving methods. Note that this algorithm does not necessarily need to be implemented by a human or a computer. The term PPES refers to the algorithm itself.
- **Computer Algebra System (CAS):** A software system that has the capability of performing symbolic manipulations (as opposed to numerical calculations) such as simplifying expressions, solving equations, and

solving indefinite integrals.

- **Intelligent Tutoring System (ITS):** A software system used for teaching a particular subject that uses student responses to create a model of the student's understanding of that subject. The ITS then uses this model to perform actions such as assigning appropriate problems or providing explanations.

1.11 Summary

This paper seeks to create a conceptual framework for use in the education of equation solving. It aims to be used by education researchers and software developers to provide a language to describe the ways in which equations are solved. After covering relevant literature, this paper describes the beginnings of a conceptual framework. This starts by introducing aspects of a meta-language of algebra and definitions. It then proceeds to describe concepts related to PPES algorithms and PPES output structures. Once this rudimentary theoretical framework has been described, the paper then covers a way to utilize this framework to perform basic experiments. The paper finishes by describing the shortcomings of this paper and future research opportunities.

2 Literature Review

This chapter covers the relevant literature related to the study and applications of PPES algorithms. The systematic study of PPESs is relatively limited due to the specialized knowledge required to formalize equation solving. The most poignant literature on the subject of PPESs comes from the artificial intelligence researcher Alan Bundy and his associates, and it is covered in great detail. Literature covering some of the most potent applications of PPES research is also covered. These sections cover two of the most interesting applications of this research: PPES comparison and intelligent tutoring system creation. The case is made that these two applications are both feasible and useful. This chapter, along with the appendixes, lays the groundwork for the conceptual framework.

2.1 Meta-Level Inference for PPESs

This section covers the history of the first PPES research as well as outlining some of its later developments. It starts first with an in depth coverage of Alan Bundy's artificial PPES called PRESS along with a history of later developments from Bundy and his associates. This is followed by a description of the operation of PRESS. This description forms the conceptual foundation for the conceptual framework presented in this paper. Lastly, some attention is paid to the minimal later research and development efforts in the area of PPESs.

2.1.1 History of PRESS

Of central importance to the current paper is the research performed at The University of Edinburgh between the early 1970s and the late 1980s because it provides the principles upon which the measurement of step-by-step equation solutions is based. This research focused on modeling human equation solving using methods from classical artificial intelligence. This research was performed by Dr. Alan Bundy and his associates. This paper abbreviates Psychologically Plausible Equation Solvers as PSES. PSES refers to any algorithm for equation solving that attempts to mimic the methods humans typically use to solve equations. Bundy started his research into automatic PSESs as part of his research into meta-level inference for use in artificial intelligence. Bundy's initial belief was that such a widely taught subject as equation solving would have well defined strategies for performing each step in the solution. However, while it was true that the axioms and rules of algebra were well established, the strategies for selecting the correct rule applied to the correct axiom at the correct point in the solution were not. This led Bundy and other researchers at The University of Edinburgh to spend over a decade identifying these strategies and encoding them into software.

In his first paper on the subject, Bundy details an explicit method for human like equation solving [Bundy 1975]. The main motivations for conducting this research, beyond advancing meta-level inference, was the apparent lack of search exhibited by humans solving equations. Bundy observed that humans do not use any of the specialized techniques used by automatic theorem provers

such as normal forms and specialized data structures, but they still produce solutions that are direct and efficient. Bundy therefore aimed to create a meta-level inference equation solver to mimic this non-specialized directness. In his initial paper, Bundy describes the meta-level goal of algebraic equation solving. He also created his core meta-level strategies for equation solving. There is no research suggesting these meta-level principles are the only way humans can solve equations. However, these principles are the only explicit theory for how humans solve equations, and they can be used to teach humans equation solving. Since these meta-level principles are the only teachable theory of human equation solving, they will motivate the investigation for much of the rest of this paper.

After identifying meta-level principles and strategies that could theoretically mimic human equation solving, Bundy and his associates at The University of Edinburgh created the first automatic PPES called PRESS (PRolog Equation Solving System) [Bundy 1981]. PRESS implemented the meta-level principles that were detailed in Bundy's 1975 paper using the programming language Prolog. It should be noted that Prolog was also created at The University of Edinburgh as part of their artificial intelligence research. PRESS uses pattern matching and rewrite rules to transform an equation into a solved form. It organizes these rewrite rules into meta-level strategies which are attempted if certain meta-level properties of the equation being solved are detected. While PRESS is capable of solving a wide variety of equations, the logical completeness of the solutions suffers from the fact that commutativity

and associativity rules are used automatically and are therefore not recorded as part of the solution. This paper uses a PPES that is heavily based on PRESS called Presston. As of the writing of this paper, Presston does not contain all of the capabilities of PRESS. However, Presston is designed so that it can explicitly show commutation and association transformations and thus seeks to be more explicit than PRESS, which is useful for educational purposes.

Shortly after the release of PRESS, the researchers at Edinburgh started work on IMPRESS (Inferring Meta-knowledge about PRESS) [Sterling L and Bundy A 1981] which is a meta-theorem prover that uses meta-meta-rules to prove the correctness of newly created meta-rules that were added to PRESS. IMPRESS was created in order to support the ability of PRESS to automatically add new meta-rules by analyzing example equation solutions. It was an essential part of creating an improved version of PRESS called LP (Learning PRESS) [Silver 1986]. This improved version of PRESS was capable of learning new meta-rules from a single step-by-step equation solution. LP did this by detecting which known rule was applied at a given step and then analyzing any steps it did not recognize. While Presston does not yet implement any aspect of LP, it may be useful in the future to implement aspects of LP in order to improve the measurement programs that are related to the conceptual framework proposed in this paper.

2.1.2 The Implementation of PRESS

Like all automatic PPESs, PRESS is designed to create a step-by-step solution

to a given equation. Each step of the solution will be some logical statement that is logically equivalent (\Leftrightarrow) to the original equation. The original equation also contains a variable that is designated as the “unknown”.

PRESS consists of a theorem prover running in proof checker mode, an object-level and a meta-level. On the object-level there are rewrite rules [Bundy 1983], which are the laws of elementary algebra combined with an inference rule named "substitution" [Bundy 1975, Gries D Schneider 1993]. The solutions created by PRESS are essentially proofs made by this theorem prover using these rewrite rules. This means that, regardless of what steps the theorem prover is directed to take, the result is logically correct. At the meta-level there are a number of strategies that are selected and attempted depending on certain meta-level characteristics of the current form of the equation. These meta-level strategies apply the theorem prover at the object-level to take the next step in the solution.

2.1.2.1 Goal of Equation Solving

In order to understand PRESS, and the conceptual framework in this paper, Bundy’s meta-level definition of a solved equation must be understood. If X is the unknown and S is some expression that does not contain X then:

1. $X=S$ is a non-trivial solution for X .
2. If A and B are statements that are non-trivial solutions for X , then $A \vee B$ is a non-trivial solution for X .
3. If $A(C)$ is a non-trivial solution for X and D is a set of real numbers,

then $\exists C \in D(A(C))$ is a non-trivial solution for X .

4. The truth values *True* and *False* are trivial solutions for X .

5. If A is a trivial or non-trivial solution for X , then A is a solution for X .

This definition of a solved equation is fundamental. It reveals the goal of all equation solving, and it will make the following strategies easier to understand. It is also crucial for identifying the necessary constraints of PPEs.

2.1.2.2 Meta-Level Strategies of PRESS

PRESS has three basic strategies at the meta-level that are each used to select the next step. These strategies attempt a new step when certain meta-level structures/patterns are present in the last step. Each strategy is associated with a set of rewrite rules that produce its intended syntactic effect. It is easiest to see the operation of PRESS by studying an example where x is being solved for:

- 1) $\ln(x+1) + \ln(x-1) = 3$
- 2) $\ln((x+1)(x-1)) = 3$
- 3) $\ln(x^2 - 1) = 3$
- 4) $x^2 - 1 = e^3$
- 5) $x^2 = e^3 + 1$
- 6) $(x = \sqrt{e^3 + 1}) \vee (x = -\sqrt{e^3 + 1})$

Figure 2: Step by Step Solution

2.1.2.2.1 Isolation

The strategy called "Isolation" is attempted when there is only one

occurrence of the unknown in an equation. Its syntactic effect is to reduce the depth of the single occurrence of the unknown. In the expression tree for line 3, there is only a single occurrence of x :

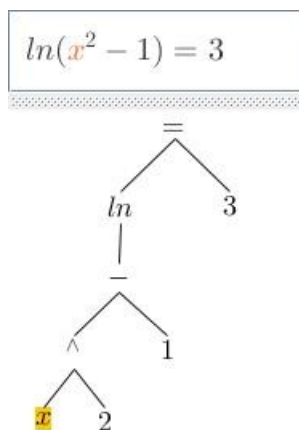


Figure 3: Line 3

The isolation rules essentially “unbury” the unknown by applying the inverse functions and operations to the dominant operator of the side of the equation that contains the unknown. For example \ln is the dominant function on the left side of the equals sign on line 3. The inverse, exponentiation of e , of \ln is then applied on both sides of the equation and then is simplified resulting in line 4.

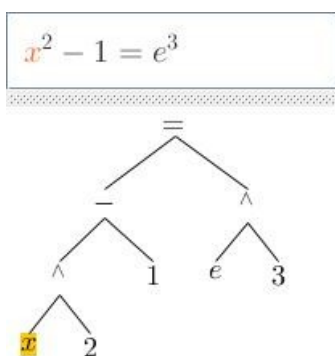


Figure 4: Line 4

Note that the unknown is now higher in line 4 than it was in line 3.

Decreasing the depth of a single instance of an unknown is the intended result of applying the isolation strategy. Since line 4 also has only one instance of the unknown, the isolation strategy can and is applied again resulting in line 5:

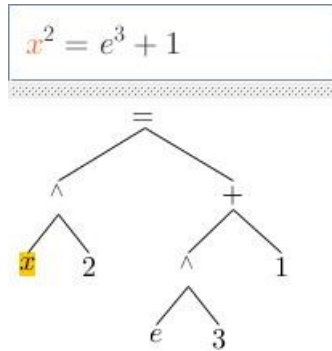


Figure 5: Line 5

Again the inverse of the dominant operator, in this case addition, is applied and the expression is simplified, and again the depth of the single unknown is reduced. Finally, another isolation rule is pattern matched and applied and we get the statement that is a solution of x :

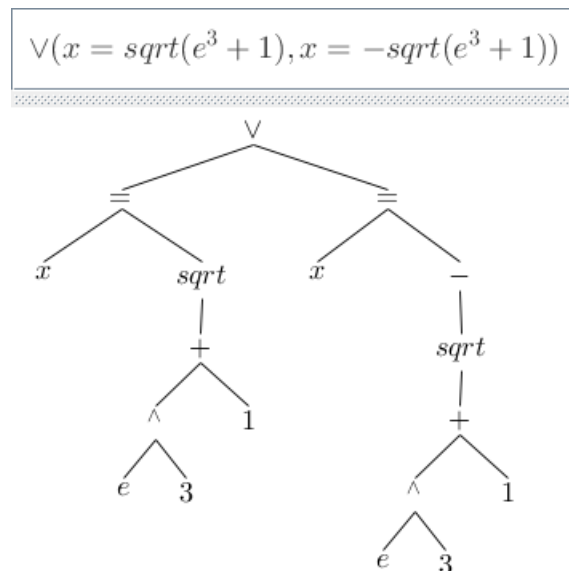


Figure 6: Line 6

Note that each of the equations under the \vee has the unknown on the left

side by itself and that the unknown does not appear on the right side of either equation. This means that they are solved, and thus the whole expression is solved.

Every rule in the theorem prover that the isolation selects has the effect of moving the equation toward a solved state by getting the unknown by itself on one side of the equation. This, however, only works if there is only one occurrence of the unknown.

2.1.2.2.2 Collection

What if there are two or more occurrences of the unknown? Then PRESS attempts to use a strategy called "Collection". The collection strategy selects rules that have the syntactic effect of reducing the total number of occurrences of the unknown. It attempts to do this until either there is only one occurrence of the unknown or it does not have any rules it can use on the equation. This is the strategy used to get from line 2 to line 3:

$$\ln((x + 1) * (x - 1)) = 3$$

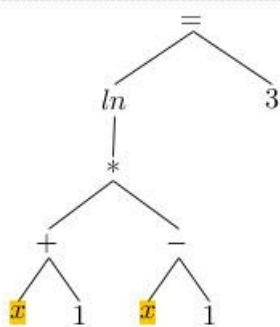


Figure 8: Line 2

$$\ln(x^2 - 1) = 3$$

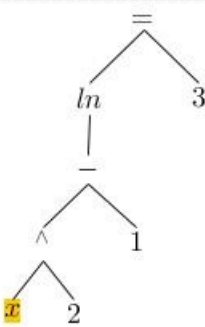


Figure 7: Line 3

Note that the collection strategy reduces the number of unknowns from two

to one. This then allows the isolation strategy to select rules that have been covered above and solve the equation.

2.1.2.2.3 Attraction

Sometimes the collection strategy is unable to reduce the number of occurrences of the unknown in an equation because their positions in the tree do not match any of the collection rules. This can be seen in line 1:

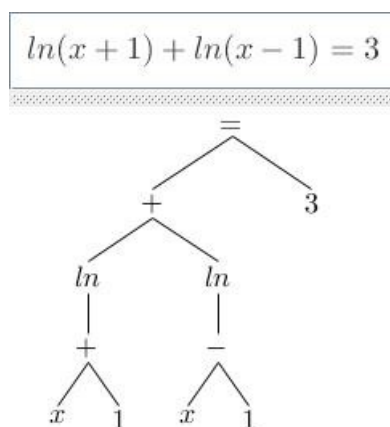


Figure 9: Line 1

There is no single collection rule that can be applied to any part of line 1 to reduce the number of unknowns. As will be shown in chapter 3, collection rules must always be used to solve equations with more than one occurrence of the unknown. There then needs to be some preparatory manipulation before a collection rule can be applied. The main preparatory strategy in PRESS is called "Attraction". With the attraction strategy, rules are applied that reduce the distance, in terms of the expression tree, of the unknowns from each other. Distance is defined in Appendix B: The Language of Expression Trees. Note that the total distance between the unknowns is reduced from 6 to 4. This is the

objective of the attraction strategy. Theoretically, as the unknowns become closer together, the likelihood that a collection rule can be applied increases.

2.1.2.2.4 The Basic Method

Taking Isolation, Collection and Attraction together into one strategy, we get what Bundy calls the basic method [Bundy 1975]. This is the fundamental strategy of PRESS. PRESS attempts to use Isolation if there is only one occurrence of the unknown, and if that fails it searches for opportunities to apply collection rules. If it cannot apply collection rules, it starts looking for opportunities to apply attraction rules to prepare for the use of the collection strategy. While these are not the only strategies in PRESS, they were the first ones to be implemented. They will help inform the concepts that are developed in the proposed conceptual framework.

2.1.3 History of Other PPESs

There have been other PPSEs that have been created since PRESS. However, only one is known to have used any of the research from Edinburgh in its creation. The others are either closed source, and thus cannot be inspected, or are open source, and do not explicitly use meta-level principles. While most of the PPESs in the following table are not documented well enough to cover in this section, MathXepert and MathSteps are:

System	Open Source	Based on PRESS	Reference
MathXpert	No	Yes	helpwithmath.com
MathSteps	Yes	No	github.com/ socraticorg/mathsteps
MathPapa	No	?	mathpapa.com
Wolfram Alpha	No	?	wolframalpha.com
Symbolab	No	?	symbolab.com
Presston	Yes	Yes	mathpiper.org

Table 2: List of PPEs

The only reliable way to characterize closed source systems that do not have documentation on their implementation is to analyze the relationship between their inputs and corresponding outputs. This type of analysis is one of the main focuses of this paper.

2.1.3.1 MathXpert

MathXpert is of interest within the current discussion because it is the only PPE that is known to have used some of the research performed by Bundy and his colleagues. MathXpert was written by Dr. Micheal Beason at San Jose State University [Beason 1996], and it is capable of creating solutions for problems in algebra, trigonometry and calculus. It is explicitly designed for education. While the software is closed source, Beason provides some insight into the construction of MathXpert in one of his papers. Beason starts the paper by identifying eight principles that PPEs should follow. These are:

1. **Cognitive fidelity:** Creates solutions that are psychologically plausible.
2. **Glass box:** Every step taken by the system can be inspected by the user.

3. **Customized to the level of the user:** Details about solutions can be hidden for advanced users.
4. **Correctness:** When producing solutions, they are logically correct.
5. **User is in control:** The user manually can take steps when solving a problem.
6. **The computer can take over:** If the user gets stuck, the computer can provide hints or worked solutions.
7. **Ease of use:** The application is easy to use.
8. **Usable with standard curriculum:** The application does not require large deviations from the standard mathematics curriculum.

Beason then describes how MathXpert follows these principles in its design. Unlike PRESS, MathXpert supports many different problem types beyond equation solving. These include common fractions, indefinite integrals and even geometry problems. In order to maintain logical correctness, MathXpert has over 1000 different logically correct operations that it uses in the creation of problem solutions. These operations are used by a theorem prover to ensure logical correctness. Unlike PRESS, MathXpert does not exclusively use rewrite rules and instead uses more general functions. Beason's reason for not using rewrite rules exclusively is his belief that they are not flexible enough. Beason used concepts from Bundy's research such as collection and attraction to prevent unwanted loops in MathXpert. Beason strove to produce problem solving strategies that were short and tidy. While Beason's outline of MathXpert's design principles

provides some insight into its operation, it is insufficient to make precise statements about its strengths and weaknesses. One of the applications of this paper would be the ability to describe the precise operation of systems like MathXpert both in general and how well they adhere to Bundy's meta principles.

2.1.3.2 MathSteps

While most PPEs are closed source, MathSteps is a notable exception. The development of this project is more ad hoc in nature, and it does not make use of any research that has been done on equation solving. Like with many open source projects, an interested developer will determine when new capabilities are required and then start adding code until the problem has been solved. The results of this paper could also be applied to MathSteps to determine the effectiveness of its design strategy.

2.2 Comparing CASs

Evidence for the usefulness of measurement programs can be found in Dr. Michael Wester's research on measuring the strengths and weaknesses of computer algebra systems (CAS) [Wester 1999]. This evidence is made stronger by the fact that automatic PPEs are specialized CASs designed to solve equations in a human-like manner. Wester's method of measuring the strengths and weaknesses of a given CAS starts by gathering together a large number of mathematics problems that should ideally be solvable by all CASs. These problems fall into a wide variety of categories such as indefinite integrals, equation solving, factoring, simplifying and other symbolic operations. This

forms what is called the "Wester test suite". The problems in this test suite are entered into a given CAS, and the results are recorded. These results are then placed into various classifications by a person depending on their simplicity and correctness. Once two or more CASs have been measured in this way, their relative strengths and weaknesses can be compared. A similar process could be preformed with the measurement programs that were developed for this paper.

Even though Wester's system of comparison is relatively simple, it has been a standard way to compare the strengths and weaknesses of various CASs. This indicates that the comparison system being developed in this paper might be similarly useful. While this paper's goal and Wester's goal of creating measurement programs for symbolic manipulation systems is similar, the implementation of Wester's measurement program and the measurement programs developed for this paper are very different. Wester's program only considers the final answers produced by the CASs, but this paper's programs primarily deal with the form of the step-by-step solutions. Further, Wester's program requires a human to categorize the final answers whereas the programs in this paper will be completely automated. Also, they will use larger datasets on a single problem type of equation solving. This paper's measurement programs also use a combination of statistics for general measurement and precise modeling for determining the adherence of a PPES to meta-level principles. Therefore, they will be more sophisticated and thus at least as useful as Wester's framework for general purpose CASs.

2.3 Intelligent Tutoring Systems (ITS)

One of the potentially potent future applications of this research is the creation of software capable of diagnosing a student's misconceptions about equation solving. Software that is capable of modeling a student's understanding of a subject, and using that model to improve said understanding, is called an intelligent tutoring system (ITS) [VanLehn 1988]. In order to improve the student's understanding of a subject area, the ITS will present problems to the student from that subject area. Using the responses given by the student, the ITS will update a model of the concepts understood by that student. If the student's understanding is inadequate, the ITS will provide some form of remedial aid to them.

An example of the power that ITSs have to significantly improve education is seen in the research into BUGGY [Brown JS and Burton RR 1978]. BUGGY was an ITS designed to diagnose misconceptions in students who were learning subtraction. In the initial research, teachers reported that many of their students appeared to be making random errors while solving arithmetic problems. The teachers assumed that their students were having difficulty following directions. However, the researchers who created BUGGY noted that there were consistencies in the errors made by individual students. They hypothesized that the student's were actually following a system of rules for creating their solutions, but some of these rules were incorrect or "buggy".

BUGGY is designed to create and modify a model of a given student's system of rules for solving subtraction problems. BUGGY initially contains a

model of an expert student as well as 330 incorrect or “buggy” rules. When a student makes mistakes while doing subtraction problems, BUGGY determines if these mistakes can be replicated using one or more of the “buggy” rules. In applying BUGGY to 1325 Nicaraguan students in the fourth, fifth and sixth grades who solved 15 subtraction problems, BUGGY hypothesized that the reason approximately 40% of the students who had made significant errors was that they were using “buggy” rules.

A similar ITS called LMS was developed by Dr. D. Sleeman [Sleeman 1984]. The LMS system uses ordered collections of rewrite rules, similar to PRESS and Presston, to model students solving equations. Ordered collections of rewrite rules are also called “production systems”. Sleeman determined that there are two possible sources of errors in student models. First is the inclusion of “buggy” rules in the production system and second is the misordering of rewrite rules in the production system, even if they are logically correct. Although Sleeman does not reference the research performed at Edinburgh or any aspect of meta-level language, these two sources of error correspond with errors at the object-level and errors at the meta-level respectively. Logically incorrect “buggy” rules correspond to errors at the object-level, and incorrect ordering of the rules corresponds with errors in the meta-level. Each production system that models a student’s understanding of equation solving consists of an ordered list of logically correct and incorrect or “buggy” rewrite rules. Due to the factorial explosion of different models, LMS tests the student’s understanding of different rules by using different problem types. Each problem type only tests the

understanding of a handful of rules at a time. Thus LMS only hypothesizes about small portions of a student's understanding at any given time which reduces the number of models considered. The framework along with the measurement programs in this paper could be used to improve the design of an ITS similar to LMS by using them to make more intelligent hypotheses about student models.

3 The Theoretical Framework

This chapter forms the central focus of this paper. It presents an outline of a conceptual framework for equation solving. This framework is not complete, but it presents an improvement in the understanding of equation solving. This framework is broken down into three parts. The first part consists of the creation of a meta-language for algebra and equation solving. This part includes the knowledge in the appendixes and the definitions in this chapter. The second part consists of a coverage of the algorithmic attributes of PPESs. This includes an attempt to identify different categories of PPESs. The third part consists of categorizing and describing the different structures and substructure that a PPES might output when solving an equation.

3.1 Preliminary Definitions and Facts

The information from all appendixes is assumed in the rest of this chapter. English will often be used as the metalanguage instead of meta-level symbols for readability. For example, this paper will often say ‘“(1·x)·5” is a product’ instead of ‘*PRODUCT*“(1·x)·5”’.

3.1.1 Definition of a Solved Equation

Bundy’s meta-level definition of a solved equation is again repeated here because it is universal. If X is a variable symbol in the object language, referred to as the unknown, and S is some expression that does not contain X then:

1. $X="S$ is a non-trivial solution for X .

2. If A and B are statements that are non-trivial solutions for X , then $A \vee B$ is a non-trivial solution for X .
3. If C is an object-level variable symbol that is different from X and $A(C)$ is a non-trivial solution for X and D is an expression that represents an object-level set of real numbers, then $\exists C \in D (A(C))$ is a non-trivial solution for X .
4. The object-level truth values "true" and "false" are trivial solutions for X .
5. If A is a trivial or non-trivial solution for X , then A is a solution for X .

3.1.2 Definition of an Equation Step-by-Step Solution

A meta-level tuple T is said to be a step-by-step solution to an equation E for some unknown X , which is a variable symbol from the object language, if T satisfies the following:

1. $T_1 = E$.
2. For every T_i , T_i is logically equivalent to E at the object-level.
3. The last member of T must be a solved equation for the unknown X .

Since it is a common practice by both human and computer PPEs to omit steps, this definition of a step-by-step solution of an equation accounts for this possibility. This means that any given intermediate step could have been arrived at using multiple implied manipulations. This also means that a tuple containing

the original equation and a final solution as its only entries in that order would be considered a step-by-step solution of the original equation solved for the unknown.

3.2 *PPES Algorithmic Facts and Attributes*

The properties discussed in this section are those that apply to the PPES algorithm itself. These will generally be inaccessible in practice since most PPES implementations are either closed source or subconscious. This section first discusses how this paper will model the algorithms of PPESs using production systems. It then covers various types of rewrite rules that would have to be present in a PPES as well as various ways these rules could be employed.

3.2.1 An Outline of a Model for PPESs

3.2.1.1 *Algorithms*

The algorithm of a PPES could be expressed using a variety of formats. It could be expressed as a collection of rewrite rules placed into strategies that are attempted based on meta-level characteristics of the most recent step, as is done in PRESS and Presston. A PPES could also use a more traditional procedural format, as it done in MathSteps and MathXpert. However, in order to ease the conceptualization of PPESs it is helpful to select a single representation that can be assumed to be the way that the PPES is implemented. This will allow assumptions to be made about the nature of PPESs that can then be used to conceptualize them.

Meta-level inference combined with rewrite rules has been selected as the

representation of PPES algorithms in this paper. This representation is the most researched for use with PPEs and it provides a well defined distinction between the object-level domain knowledge and the meta-level control knowledge [Van Harmelen 1991]. By using this representation of PPEs, concepts created during the research performed at Edinburgh can be incorporated more easily into this paper's conceptual framework. By separating the object-level domain knowledge from the meta-level control knowledge they can be conceptualized separately. This is useful because the domain knowledge of algebra is well established and the control knowledge can then be studied with more clarity. Since rewrite systems are Turing complete [Godoy, G., & Tiwari, A. 2005], they can simulate any other implementation of a PPES algorithm. Since the rewrite rule system can emulate any other implementation, the format in which the PPES was first implemented is irrelevant.

3.2.1.2 Meta-Level Control

Meta-level control in a rewrite rule system can be engineered in several ways. Some of the main methods include the following:

1. The order in which the rewrite rules are placed in the rewrite rule system can be changed.
2. Preconditions can be placed on the rewrite rules.
3. The order in which rule pattern matching is done can be changed.

3.2.1.3 Categorizing Rewrite Rules

The rewrite rules themselves can each be categorized several different ways. Here are some distinctions that can be made between algebraic rewrite rules using the meta-level language:

1. A rule can either rewrite only a subtree of an equation or the whole equation.
 - a) If the rule only rewrites a subtree of an equation, then that rewrite rule will generally be based on an axiom or theorem which is a universally quantified equation.
 - b) If the rule rewrites the whole equation then it will either 1) completely replace the equation with a disjunction of equations or an existentially quantified equation or 2) use the “whatever you do to one side of an equation you can do to the other” principle.
2. As Bundy’s conceptualization of PPESs shows, a rewrite rule might, or might not, have one or more of the following effects. Note that the meta-level effect of a rewrite rule is dependent on the preconditions placed on that rule by the meta-level control system that is applying the rule:
 - a) The rule may reduce the depth of a subtree of an expression.
 - b) The rule may reduce the number of instances of a particular subtree in an expression.
 - c) The rule may bring two subtrees closer together within an expression.

3.2.1.4 Sameness Through Transformation

Another concept that should be considered is the notion that a part of an expression is the “same” before and after a rewrite. It is often stated that some subtree of an expression that has been rewritten using a rewrite rule is the same before and after the rewrite. This can be formalized by considering any subtree that was matched to a pattern variable on the left side of a rewrite rule to be the same if that variable appears on the right hand side of the rewrite rule.

3.2.2 The Necessity of Depth Reducing, Collection and Pre-Collection Rules

Some generalizations can be made about the nature of a PPES algorithm. Among these are generalizations about the necessary meta-level effects that must be brought about by some applications of the rewrite rules in the PPES algorithm. The generalizations stated in this section are based on the above definition of a solved equation, particularly the fact that there is only one instance of the unknown that is at a depth of one below the equals sign in the tree of a solved equation. If the starting equation has more than one instance of the unknown, then rules must be applied that have the effect of reducing the number of instances of the unknown to one. If the equation ever has only one instance of the unknown that is not a child of the equals sign, then the depth of that instance must be reduced so that it is directly under the equals sign. It should be noted that the following types of rules, along with meta-level controls, could also be used to achieve meta-level goals in algebra other than equation solving such as simplification or factoring.

3.2.2.1 Depth Reducing Rules

Depth reducing rules can be attempted on any single subtree within an expression. The subtree that is to have its depth reduced shall be called the targeted subtree. The effect of these rewrite rules, along with meta-level control, is to reduce the depth of the targeted subtree. There are several different ways that the depth of a subtree can be reduced:

1. A rewrite rule that uses the fact that “whatever you do to one side of an equation can be done to the other” to apply an inverse function of the dominate operator of the side that contains the targeted subtree. This method is a generalization of Bundy’s isolation strategy. It will always be an option since every function has an inverse relation that can have zero or more possible outputs for every input. This kind of depth reduction can be done in one or more steps, depending on the construction of the PPES. The first step could introduce the inverse relation and the second step could use whatever cancellation axiom or theorem that is related to that relation to reduce the depth of the targeted subtree. Note that this kind of depth reduction can result in the equation being rewritten into a disjunction of equations, like when eliminating $\sqrt{\quad}$, or an existential statement containing an equation, like when eliminating \sin . It can also result in a contradiction. Note that if this process is done in two steps, the depth of the targeted subtree may be increased temporarily by the first step before the second step reduces it.
2. A rewrite rule that only rewrites part of the expression could also be used

to reduce the depth of a targeted subtree. For example, the associative property in rewrite rule form can be used to reduce the depth of the targeted subtree.

There are several strategies that a PPES can use with depth reducing rewrite rules to bring an equation closer to a solved state.

1. If the most recent step has only one instance of the unknown and that instance is not at depth one as a direct child of the equals sign, then rewrite rules that reduce the depth of this single instance of the unknown must be used to reach a solved equation. This strategy will be called the "finishing depth-reduction" strategy.
2. If there is more than one instance of the unknown and all the instances of the unknown are on one side of the equation, then the smallest subtree containing all of the unknowns can have its depth reduced using the above methods. By the smallest subtree is meant the smallest subtree in the current expression. It is not the smallest possible subtree that could be reached through manipulation. This would generally result in this subtree being a direct child of the equals sign. Other rewrite rules would then be used to reduce the number of instances of the unknown to one. If at any point during this collection process the current smallest subtree containing all of the instances of the unknown is not at depth one, then another depth reducing rewrite rule can be applied to this new subtree. This strategy will be called the "opportunistic depth reduction" strategy.

3. If there is more than one instance of the unknown and all instances of the unknown are on one side of the equation, then the PPES can apply depth reducing rewrite rules to the smallest subtree containing all of the instances of the unknown, interspersed with other rewrite rules. This will be called the "mixed depth reduction" strategy.

3.2.2.2 Collection Rules

If there are two or more instances of a subtree in an expression that are identical, then a collection rewrite rule along with meta-level control can be attempted on these subtrees in order to reduce the total number of instances of them in the entire expression. Note that all instances of the subtree to be collected will have to be on the same side of an equation in order to be collected.

A PPES can use these types of rewrite rules in a few ways:

1. The rules can be attempted directly on the instances of the unknown themselves. This is Bundy's collection strategy
2. The rules can be attempted on identical instances of subtrees which contain instances of the unknown. There are methods that will be outlined in the next section that prepare the tree for the use of this type of rule.

This method is also used by Bundy in PRESS.

3.2.2.3 Pre-Collection Rules

Pre-collection rules are used to rewrite an expression into a form in which collection rules can be successfully applied. There are several different forms of

these pre-collection rules:

1. Attraction rules are those rewrite rules that bring two or more subtrees closer by reducing the number of arcs that must be traversed in order to reach every subtree. Since the patterns on the left side of every collection rewrite rule only have a limited size, reducing the size of the smallest subtree containing all the instances of the subtree we are trying to apply collection to increases the chances that a collection rule will match. This is essentially Bundy's attraction strategy.
2. The "homogenization pre-collection" strategy. This strategy searches for ways to change different subtrees that contain instances of the subtree that is to be collected into an identical form so that collection can be performed on them. This is a generalization of Bundy's homogenization strategy.
3. "Specialized pre-collection" rules are each only used to prepare for a specific collection rule. For example, if bag structures are not used in the PPES, factoring out two instance of the unknown may require a specific ordering to be used, thus a specialized pre-collection rule may be applied to achieve this reordering. Note that, due to the less explicit nature of PRESS, these specialized rewrite rules do not appear in it.

3.3 PPES Output Facts and Meta-Level Functions and Predicates

This section identifies and explores various meta-level functions and predicates that take meta-level structures found in the step-by-step solutions

produced by PPEs and return useful information. This section has been separated into five tiers of meta-functions. Each tier contains functions that take structures of a certain type. Each tier uses the structures from the lower tiers to define the structures at its own tier. All tiers of meta-functions are ultimately based on the primitive concepts discussed in the appendices.

3.3.1 Sub-Equation Meta-Level Functions and Predicates

Sub-equation meta-functions are those functions that take as their inputs expressions that represent numbers. Thus any expression that contains logical or set theory symbols will not be considered in this section. All of the attributes in this section will be about subtrees.

The following list are predicates and functions that can be applied to expressions that represent numbers:

- Sum Predicate
- Addends of Sum Function
- Product Predicate
- Factors of Product Function
- Polynomial Predicate
- Rational Predicate
- Transcendental Predicate

3.3.2 Single-Step Meta-Level Functions and Predicates

Single-step meta-level functions and predicates take a single entry in a step-by-step solution of an equation as their inputs. In general, they will apply to a single equation. However, when a step is more complex, they could apply to each equation that is present in that single step such as when the step is a disjunction.

3.3.2.1 *Instances of the Unknown Meta-Level Function*

When solving an equation, the configuration of all the instances of the unknown is very important. Since the equation must be rewritten into a form that has only one instance of the unknown by itself as a child of the equals sign, identifying if a rewrite brings the equation closer to this final state allows for proper meta-level control.

3.3.2.1.1 Positions of Unknowns Function

This function takes an equation as an input and outputs a set of tuples that represent the positions of all of the instances of the unknown in the equation tree. This function returns the most information of all the functions related to instances of the unknown, and all the other unknown related functions can be based on this function.

3.3.2.1.2 Number of Unknowns Function

This function takes a single equation as its input and outputs a non-negative integer which represents the number of instances of the unknown. This function is important for measuring the success of collection rewrite rules.

3.3.2.1.3 Smallest Unknown-Containing Subtree Function

This function takes a single equation and returns the position where the smallest subtree containing all the instances of the unknown is. If instances of the unknown appear on both sides of the equation, then the function will return an empty tuple that indicates the whole tree is the smallest subtree which contains all of the instances of the unknown. This function can be useful when answering questions about whether a PPES is using an opportunistic depth-reduction strategy or a mixed depth-reduction strategy.

3.3.2.1.4 Maximum Depth of Unknown Function

This function takes a single equation as an input and returns a positive integer which is the maximum depth at which an instance of the unknown appears. Since a solved equation must have the unknown as one of the children of the equals sign, measuring the depths of the instances of the unknown can indicate how close the equation is to being solved.

3.3.2.1.5 Minimum Depth of Unknown Function

This function takes a single equation as an input and returns a positive integer which is the minimum depth at which an instance of the unknown appears in the equation.

3.3.2.1.6 Distance Between All Unknowns Function

This function takes a single equation as an input and outputs a non-negative integer which represents the total distance between all of the unknowns. Since each instance of the unknown is a leaf of the equation tree, the instances of the

unknown can be ordered from left to right. There will then be some number of arcs that connect each instance of the unknown to the next instance of the unknown. The total distance between all instances of the unknown is defined as the sum of all the lengths between these consecutive instances of the unknown.

3.3.3 Multi-Step Meta-Level Functions and Predicates

Multi-step functions are those functions that take more than one step to as input. These functions will primarily be used to identify changes made from one step to the next and thus will focus on two consecutive steps in a solution.

3.3.3.1 Consecutive Step Functions

Consecutive step functions take two consecutive steps as input. These functions can be used to provide insight into the missing information between steps if a PPES performs multiple transformations in one step.

3.3.3.1.1 Number of Unknowns Change Function

This function takes two consecutive equations in the step-by-step solution to an equation from a particular PPES and returns an integer which is the number of instances of the unknown for the second equation minus the number of instances of the unknown in the first equation. This indicates how many depth reducing rewrite rules were used in between the two steps.

3.3.3.1.2 Maximum Depth Change Function

This function takes two consecutive equations in the step-by-step solution to an equation from a particular PPES and returns an integer which is the

maximum depth of the unknown for the second equation minus the maximum depth of the unknown in the first equation. This indicates how many collection rewrite rules were used between the two steps.

3.3.3.1.3 Distance Change Function

This function takes two consecutive equations in the step-by-step solution to an equation from a particular PPES and returns an integer which is the distance between all the instances of the unknown for the second equation minus the distance between all instances of the unknown in the first equation. This indicates how many attraction steps were taken between the two steps.

3.3.4 Single-Solution Meta-Level Functions

Single-solution meta-level functions take a single step-by-step solution of an equation for some unknown.

3.3.4.1.1 Number of Steps Function

This function takes a step-by-step solution and returns the number of steps that are in that solution. This can be used to compare the level of detail provided by different PPESs as well as the level of detail between different types of problems. This property can also be used to identify relationships between the structure of the equation and the number of steps in the solution.

3.3.5 Multi-Solution Meta-Level Functions and Algorithms

This tier does not consist of proper functions in the mathematical sense, although some are included. It instead mostly consists of algorithms that help to

generate theories about a PPES whose algorithm is inaccessible. Since this is the top level-tier, the generation of the equations that are used to make these solutions must also be considered in addition to the way they are solved. There are several broad methods of making and measuring multiple step-by-step solutions.

3.3.5.1 *Random Equation and Statistics Approach*

This approach of using multiple solutions is the easiest. In it, equations are randomly generated and solved. Statistics are then collected on these step-by-step solutions by using lower tiers of meta-level functions. Of particular interest is the calculation of steps and their correlation with other numerical attributes of the initial equation. This can give some idea of what structures in the initial equation contributed to the length of solutions. The implementation of this approach in MathPiper will be covered in chapter 4.

3.3.5.2 *Pattern Recognition Approach (Not Implemented)*

A more sophisticated method would start by randomly generating a database of equations along with their solutions, but instead of selecting the statistics to measure as above, apply pattern recognition software to detect relationships between all of the initial equations and the various structures and sub-structures within their corresponding step-by-step solutions of that equation.

3.3.5.3 *Intelligent Modeling Approach (Not Implemented)*

An even more sophisticated approach would consist of using the constraints placed on acceptable PPES algorithms revealed in the above conceptual

framework to create hypotheses about the algorithm that is being used in a particular PPES. Then provide equations that test these hypotheses by predicting what the PPES might do to solve a particular equation for each hypothesis and then check if the PPES takes any of these predicted approaches.

4 Implementation of Measurement Programs in MathPiper

This chapter is meant to help future researchers and software developers to recreate some of the software that has been based on the above conceptual framework. In particular, it will provide guidance as to how a random equation and statistics approach for characterizing a PPES may be implemented in software. It will cover the choice of programming language and the reasons for making that choice. It discusses aspects of random equation generation as well as some aspects of the PPES used. It then considers a possible way to set up experiments. It finishes with the creation of a couple simple linear regression models relating numerical attributes of the starting equation to the number of steps in the step-by-step solution.

These regression models have been selected because they are related to depth reducing and attraction strategies that must be used in a PPES. They are based on 1000 randomly generated equations and their resulting solutions created using Presston. The three numerical attributes of these equations and resulting solutions that have been selected are:

1. The maximum depth for all the instances of the unknown in the starting equation.
2. The total distance between the instances of the unknown in the starting equation.
3. The number of steps in the step-by-step solution made by Presston.

4.1 Construction of Experiments

A MathPiper worksheet has been created that includes all of the procedures that are required for experimentation. The steps of this initial experimentation are:

1. Generate a list of random equations using the method described below.
Note that as part of this step, the depth of the abstract expression tree database, the number of unknowns, the range of numbers and letters to be used for leaves and the set of operators that are randomly selected from must be decided. Those equations are saved in case other researchers wish to repeat the experiment with the exact same equations.
2. A loop is run that walks through the list of randomly generated equations and uses a PPES to obtain the step-by-step solutions for these equations in the PPES. The attributes that the researcher desires to measure are then obtained from the solutions before the next equation is solved.
3. Finally, after obtaining the attributes from all of the equation solutions, statistical analysis can be preformed such as measuring proportions or linear regression between numerical attributes.

4.2 Functionality of MathPiper

MathPiper was selected as the CAS to build the prototype experimental framework for the measurement programs to be tested. MathPiper is an open source CAS being developed at Shawnee State University, which is written in the Java programming language. Built on top of this Java base is the MathPiper

language, which is a version of the Lisp programming language. The MathPiper interpreter hides the underlying Lisp syntax with a traditional infix syntax to increase readability. MathPiper is distributed with an IDE (Integrated Development Environment) called MathPiperIDE. MathPiperIDE uses plain text files to work with the MathPiper source code. The text files are given a .mpws extension and use a worksheet format like many other CASs such as Mathematica and Sage. MathPiperIDE also includes several additional applications bundled with it that allow it to output visual representations of various types.

MathPiper has several capabilities that make it suitable for experimenting with algebraic meta-level inference. The MathPiper language has been designed to have both procedural and declarative capabilities. This makes it easy to implement a PPES and then experiment on it. Since MathPiper is a CAS, it has the ability to treat symbols as their own data type and supports many symbolic manipulation techniques. MathPiper also has tree manipulation and representation capabilities. All of the diagrams of trees in this paper have been created using procedures in MathPiper. The tree manipulation procedures in MathPiper have been used to construct many of the procedures that are used in the measurement programs.

Another symbolic advantage that MathPiper has is the ability to implement systems of rewrite rules. Most of the CAS symbolic procedures in MathPiper are implemented using rewrite rules. MathPiper supports a variety of preconditions on these rewrite rules. It also supports local rewrite rule sets, which makes

experimenting with new rule sets possible for users. The prototype of a PPES called Presston has been implemented in the MathPiper language and is included with the distribution of MathPiperIDE as of the writing of this paper.

Presston is a PRESS inspired PPES that is implemented and included in MathPiper. As of the writing of this paper, Presston is still in the alpha stage of development. It uses many of the same strategies used in PRESS, although they are called by different names. Versions of the isolation, collection and attraction strategies are partially implemented in Presston. Presston deviates from PRESS by not using bag structures for its sums and products. Presston handles some of the complications that this introduces by using normal forms for sums and products. It is hoped that through the process of this research, better solutions than normal forms will be identified to handle the pre-collection portions of Presston.

4.3 Generating Random Equations

In order to make accurate and precise inferential statistical models containing attributes from a particular PPES, an understanding of random equation generation must be obtained. Of particular interest when generating random equations is the distributions of numerical and categorical attributes produced. Assumptions of any statistical models based on these attributes must be made true by the equation generation algorithm. This section demonstrates a method for generating equations that has significant shortcomings for inferential statistics and gives some insight as to how these shortcomings may be

eliminated. It is meant to outline the difficulties encountered when randomly generating equations, but it does not claim to solve these difficulties.

4.3.1 Naive Method

This section describes a method for generating random equation that will be referred to as the “naive method”. The tree capabilities in MathPiper allow a unique way of creating random equations which is used in the experiments. This method is centered on the concept of an “abstract expression tree”. An abstract expression tree contains operator meta-level symbols (represented by uppercase "O"s) and leaf node meta-level symbols (represented by uppercase "L"s) in a tree structure. These symbols are generic place holders that can later be replaced with actual object-level operator, constant and variable symbols, but the abstract expression tree does not contain any operator, constant or variable symbols itself. Examples of abstract expression trees are shown in Figure 10.

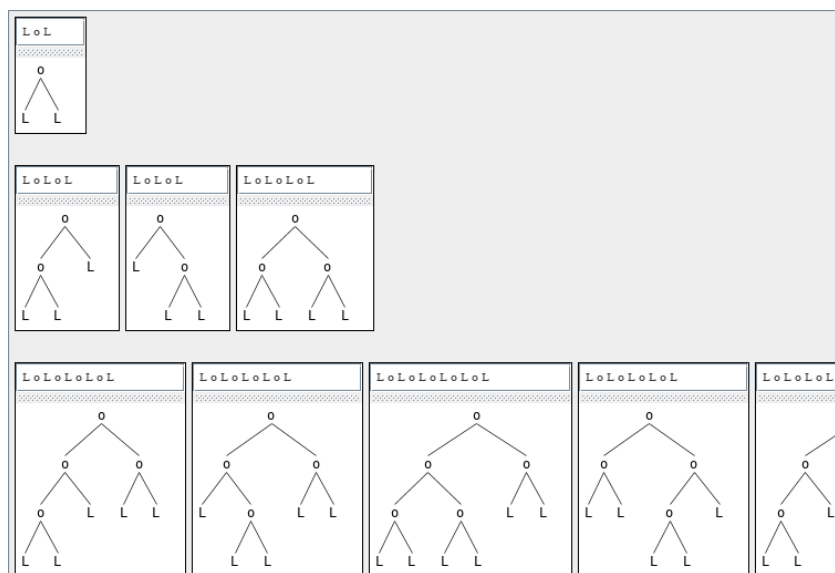


Figure 10: Abstract Expression Tree Databases

The method used for generating equations is as follows:

1. The process starts with the exhaustive creation of a database of abstract expression trees down to a specified depth. This allows control of the maximum depth of the equations generated from this method. This means that if a depth of 3 was selected during the creation of the database, every possible form of abstract expression tree that has a maximum depth of 1, 2 or 3 would exist in the database. See Figure 10.
2. A predetermined number of instances of the unknown is then selected by the researcher.
3. Once the database has been created, an abstract expression tree is randomly selected. It should be noted that each of the abstract expression trees in the database are unique and equally likely to be picked during equation generation with this method.
4. leaves from the selected abstract expression tree are randomly chosen to be changed to an instance of the unknown.
5. The other leaves are then changed into either letter constants or number constants according to the desire of the researcher.
6. Finally, the root node is changed to an equals sign and the rest of the operator symbols are each randomly changed to one of a set of accepted operator symbols. This process allows much control over the generation of random equations.

A set of 1000 random equations were created for the purposes of testing. A depth of 4 was selected for the creation of the abstract expression tree database. The number of instances of the unknown placed in each equation during generation was set to 2. The non-unknown leaves are randomly replaced by consecutive integers starting at 1. The set of operator symbols that were placed in the place holders were addition, subtraction and multiplication. The random equations that were generated have been recorded in the MathPiper worksheet for repeatably. Every set of measurements taken from a list of random equations is repeatable if these equations are saved.

The following attributes of the step-by-step solutions were measured:

1. The maximum depth of instances of the unknown in the initial equation.
2. The number of unknowns in the initial equation.
3. The distance between the unknowns in the initial equation.
4. The number of addition signs in the initial equation.
5. The number of subtraction signs in the initial equation.
6. The number of multiplication signs in the initial equation.
7. The number of steps in the step-by-step solution.
8. A list of each rule used in the order they were used by Presston.
9. A list of the meta-level strategy at each step used by Presston.

Note that the items 8 and 9 could only be measured in a PPES that provides

this information, which most do not. However, it is hoped that in the future there will be ways to deduce which rewrite rule or meta-level strategy was used at each step, and this simulates that capability.

4.3.2 Shortcomings of the Naive Method

There are two shortcomings with the naive method.

1. The distribution of numerical attributes of the generated equations, such as maximum depth of the unknown and distance between the unknowns, are not normally distributed.
2. The numerical attributes that one may wish to use as independent variables in a statistical model are moderately correlated.

The distributions of three numerical attributes of the randomly generated equations will be considered here to demonstrate the need for a better method of random equation generation. These numerical attributes are of interest because they will be used to construct a collection of regression models. These attributes are: the maximum depth of all instances of the unknown in the starting equation, the total distance of the instances of the unknown in the starting equation and the total number of steps in the Preston produced solution. A list of 1000 randomly generated equations was created using the naive method to demonstrate the shortcomings of such a method. This will motivate a discussion about how to overcome these shortcomings in the next section.

4.3.2.1 Attributes Not Normally Distributed

The descriptive statistics and histograms obtained from the numerical attributes of the 1000 randomly generated equations and their solutions are included in Table 3. It should be noted that all attributes are discrete.

	Maximum Depth	Total Distance	Step Count
Mean	3.830	5.879	19.588
Median	4	6	19
Standard Deviation	0.402	1.731	6.051
IQR	0	2	9
Range	2	2	34
Q1	4	5	15
Q3	4	7	24
Minimum	2	2	6
Maximum	4	8	40

Table 3: Descriptive Statistics

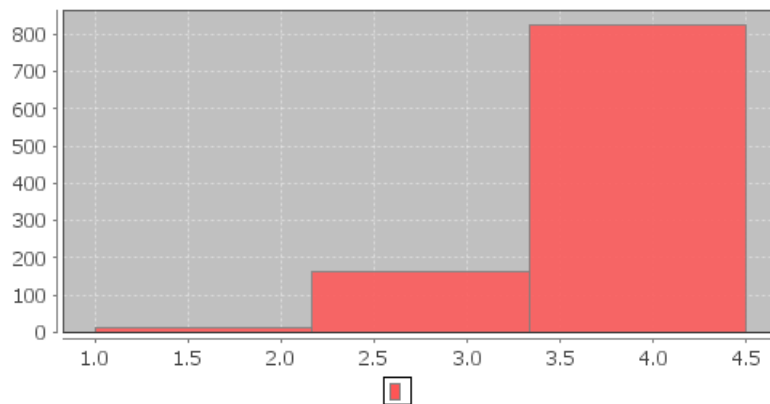


Figure 11: Starting Maximum Depth Histogram

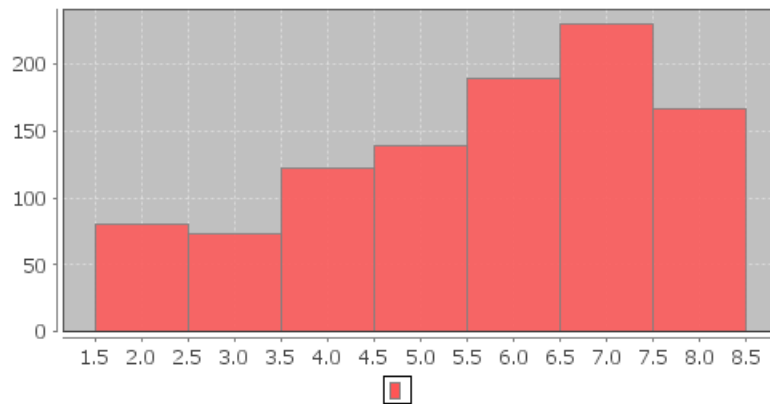


Figure 12: Starting Total Distance Histogram

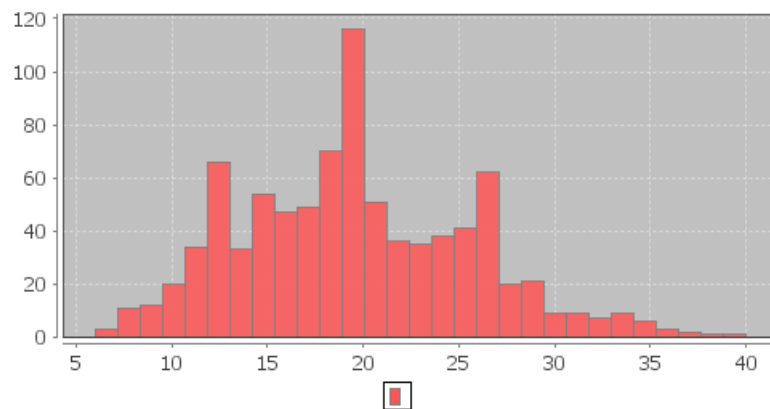


Figure 13: Step Count Histogram

The histograms in Figure 11 and Figure 12 show one of the main shortcomings of the naive method for random equation generation stated above. As can be seen in Figure 11, the distribution of the starting maximum depth of the unknown is heavily left-skewed. In addition, Figure 12 shows that the distribution for the starting total distance between all of the instances of the unknown is also heavily left-skewed. If the assumptions of a statistical model, such as the regression models shown later, state that the distributions of certain

numerical attributes must be normal, the naive method produces equations that violate these assumptions.

The fact that these distributions are skewed becomes obvious upon reflection. In the database of abstract expression trees, there are far more abstract expression trees with leaves at deeper levels than there are at shallower levels. While the functional relationship between the number of abstract expression trees at a particular maximum depth level and that depth level has not been determined, a lower bound is 2^x . Since every bottom level leaf of every abstract expression tree at a given level can be replaced with an operator symbol and since each abstract expression tree at that level has at least two such leaves, the next level must therefore have at least twice the number of abstract expression trees as the previous level.

Since each abstract expression tree in the generated database is equally likely to be selected, it is far more likely that one with a deeper maximum depth will be selected. Since each leaf of this abstract expression tree is equally likely to have an instance of the unknown placed in it, there are going to be far more equations with a maximum depth of the unknown at a deeper level. Similarly, the abstract expression trees that have a deeper maximum depth also have a proportionally wider base. This means that there are more opportunities for a random equation generated using the naive method to have unknowns further apart and thus result in a larger starting total distance between the instances of the unknown. This also results in a left-skewed distribution for the starting total distance attribute.

4.3.2.2 Undesirable Correlations of Numerical Attributes

The following is a summary of the regression model created with maximum starting depth predicting starting total distance:

Model:	$\text{Distance} = b_0 + b_1 \text{Depth}$
r	0.332
b_0	0.404
b_1	1.429

Table 4: Distance and Depth Regression

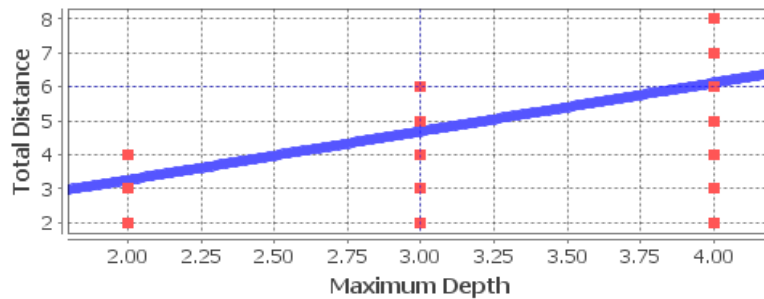


Figure 14: Depth and Distance Regression Line

While a full regression analysis has not been performed, it can be seen that there is a relationship between the two numerical attributes. This is undesirable because they are to be used as independent variables in a linear regression.

4.3.3 Ways to Improve the Naive Method

The main two problems with the naive method are that the distributions of the numerical attributes, in this case starting maximum depth and starting total distance, are not normally distributed and they are correlated. It is technically impossible for discrete distributions to follow a normal distribution, so the goals for an improved random equation generation method are:

1. Specify the discrete joint probability distribution of the constrained numerical attributes so that the individual attributes are normally distributed and are not correlated. This could be achieved practically by assigning a number between 0 and 1 to each possible combination of values for the numerical attributes whose sum is 1.
2. Create a dataset that has approximately the specified proportion of equations for each combination of the constrained numerical attributes.

There are several ways in which a dataset with the desired properties may be created. The easiest would be to create a dataset of equations using the naive method and then select a subset of those equations which approximately creates the desired proportion of equations for each combination of values from the constrained numerical attributes.

Another more sophisticated method for improving the distributions of the numerical attributes of the randomly generated equations would be to adjust the probability of generating equations with certain values for their numerical attributes. There are two places in the naive method where probabilities are assigned uniformly:

1. The selection of which abstract expression tree to generate a random equation with.
2. The selection of which leaves of the abstract expression tree will be replaced with instances of the unknown.

There are at least a few options for assigning probabilities to both of these

selections:

1. Create a finite set of mutually exclusive and exhaustive subsets of the abstract expression trees down to a specified depth, and assign probabilities to each subset. Once a subset has been randomly selected, an abstract expression tree would be selected from the subset with equal likelihood assigned to each member of the subset. The instances of the unknown would be assigned as in the naive method with equal likelihood.
2. Assign a probability to each abstract expression tree in the database individually and handle instances of the unknown as in the naive method.

4.4 Regression Models

The two simple linear regression models that have been created using these variables are:

1. Using starting maximum depth to predict step count.
2. Using starting total distance to predict step count.

A model combining depth and distance as predictors was not created because of the shortcomings of the random equation generation method which would invalidate any further work performed on such a model. While the random equation generation method invalidates these models, they have been included for demonstrating the possible statistical models that can be created for a PPES.

Model:	$\hat{\text{Steps}} = b_0 + b_1 \text{Depth}$	$\hat{\text{Steps}} = b_0 + b_1 \text{Distance}$
r	0.365	0.663
b_0	-1.428	5.960
b_1	5.487	2.319

Table 5: Regression Models

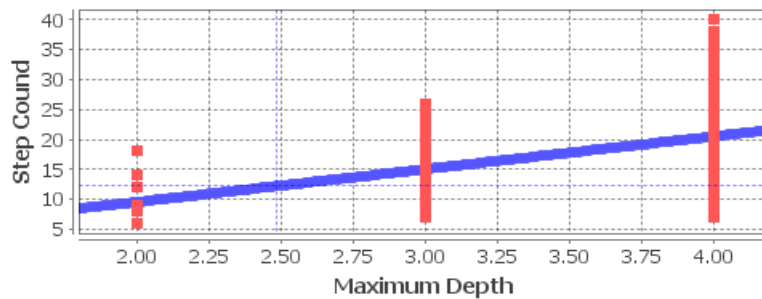


Figure 15: Depth and Steps Regression Line

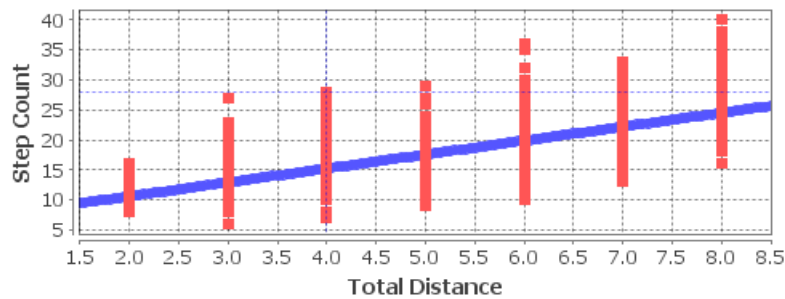


Figure 16: Distance and Steps Regression Line

If the PPES that a linear regression model comes from is both efficient and does not omit steps, then it should be expected that there will be one or two depth reducing steps for every level increase in the starting maximum depth of the unknown. If there is less than this, then steps are likely being omitted. With a slope for the first regression model, as seen in Table 4 and Figure 15, that is considerably higher, it is likely that the skewing of the depth distribution and the

correlation with the starting total distance is causing an inflated value for the slope. This may also account for the first model's relatively low correlation coefficient.

A similar, although less severe result, can be observed with the second model, as seen in Table 4 and Figure 16. Again, a single arc increase in the starting total distance between the instances of the unknown should result in at most one or two more steps in the solution. Much less than that would indicate that steps are being omitted. It is also likely that the skewed distribution and correlation with starting maximum depth is contributing to the inflated slope value for the second model. No further analysis was performed due to the bad dataset.

5 Conclusion

This chapter is meant summarize the results fro the tapper, the shortcomings of this paper and the posible future research that can be done in this area.

5.1 Summary

This paper has presented the beginnings of a conceptual framework for equation solving. This conceptual framework is far from complete, but it allows some amount of progress to be made in understanding the nature of PPEs. This conceptualization is meant to be used to improve the education of equation solving. At the most basic level of application, it is hoped that the conceptualization itself can be taught with the goal of improving the efficiency and effectiveness of learning equation solving. The conceptual framework is also meant to eventually be used by researchers who wish to characterize either the equation solving methods of different people or the equation solving methods being taught. Refined versions of this conceptualization should also help software developers to describe and develop educational software for teaching equation solving. The ultimate application of research into PPEs would be the creation of a ITS for equation solving.

5.1.1 Construction of the Conceptual Framework

The conceptual framework presented in this paper has been separated into three parts. These parts may be merged or split in the future as the research continues, but they provide an adequate structure for the time being. The first

part is the formation of a formal meta-language which provides the most basic concepts necessary to define the rest of the concepts. This part has largely been placed in the appendixes. Second is the conceptualization of the algorithmic attributes of a PPES. Third is the conceptualization of the output attributes of PPESs. These together form the conceptual framework.

The understanding that a meta-language must be used in order to precisely describe the strategies used in equation solving is one of the key insights produced by Alan Bundy from the University of Edinburgh, and this paper uses it as a foundational principle. The meta-language presented in this paper is incomplete due to research time constraints. The preliminary definitions and the contents of the appendixes, while not especially precise, provide a basis for thinking about equation solving using a formal meta-level language. The meta-level language presented in this paper is adequate for initial purposes, but it should be improved in future research.

Using this meta-language, the distinction has been made between the algorithmic attributes of a PPES and the resulting output attributes of a PPES. This distinction has been made because the algorithmic attributes of most PPES systems, either human or machine, are unavailable for inspection. Therefore, PPESs need both to be conceptualized as “black box” input-output-systems and as algorithms.

The study of algorithmic attributes presented in this paper covers some constraints on the types of algorithms that can solve equations. The method of

modeling PPESs as a rewrite-rule system with meta-level inference has also been presented in order to leverage the existing research on PPESs and provide a tested format for performing proofs in future research. This paper has focused on the core effects that the strategies used in the PPES must accomplish in order to be a working general equation solver that mimics human processes. These goals are: depth-reduction, collection and pre-collection.

This paper has also provided a five-tiered framework for categorizing the structures and sub-structures that will be outputted when a PPES algorithm is applied to an equation that has a symbol variable identified as the unknown. These tiers can be used to create formal definitions for the basic meta-level functions and predicates that apply to these structures. These predicates and functions obtain potentially useful information for researchers and software developers. While these functions and predicates are not particularly sophisticated, they indicate what is possible for future development.

5.1.2 Implementing Measurement Programs

Using the above conceptual framework, this paper has presented a pilot experiment for measuring the algorithmic attributes of a PPES using its output attributes. A simple experimental framework for doing this by generating random equations and their solutions and then performing statistical analysis on the results is presented in this paper. The main result from these experiments was the realization that the method for creating random equations used in this paper, referred to as the naive method, is inadequate for statistical modeling.

Due to this inadequacy, several suggestions have been made for improving the naive method. Despite these shortcomings, a couple of simple linear regression models have been given in order to demonstrate what is possible with this experimental framework.

Reasons have been given for the use of the MathPiper language in implementing the measurement of the output attributes of the Presston PPES. In addition to the fact that Presston is written in the MathPiper language, which allowed seamless integration of experiments with the solving code, MathPiper's capabilities have been well suited to the experiments performed. The fact that MathPiper has mathematical expressions as a fundamental datatype aided in the manipulation of equations. In particular, MathPiper's ability to use pattern matching to parse and search expression trees was helpful. This is in addition to its many visualization tools. Future researchers should consider the use of MathPiper when investigating PPESs.

A key insight provided by these experiments was the importance of engineering a method for generating random equations that creates the correct distributions of the numerical attributes of the generated equations. A method of generating random equations was tested and found inadequate. Certain numerical attributes of equations generated using the naive method were found to be heavily skewed and correlated with each other. These attributes might be useful to include in statistical models if their joint distribution were appropriate. Several suggestions for possible ways to improve the existing method have been made, although none have been attempted.

This paper has presented a rudimentary experimental framework which consists of creating random equations and solving them with a PPES in order to obtain statistics that can be used to characterize that PPES. This framework has been used in the experiments performed for this paper by encoding it into a MathPiper worksheet. While this method of gaining information is simple, it still provides some insight about the PPES such as how many steps the algorithm is omitting. The simple linear regression models produced for this paper using this experimental framework demonstrated the possibility of creating statistical models to characterize PPESs that are not inspectable. More research would need to be performed to determine what other statistics could be of use.

5.2 Shortcomings

Since this paper is intended as an entry point into the area of human-like equation solving, the coverage has notable shortcomings. Some of these shortcomings are the result of the struggle to properly conceptualize an area that only has a small amount of relatively specialized research performed on it. The research included in this paper only gives a broad outline of the topics included because they are still being conceptualized. A more precise and detailed investigation into these topics would have to be made in order to be used in future research and practical applications. The statistical analysis in particular could have been better conceptualized. These shortcomings are considered in this section

This paper sought to communicate the necessity of a formal and symbolic

coverage of the meta-level aspects of equation solving. While there are the beginnings of a meta-language capable of expressing the complexities of equation solving, a complete meta-language has not been established. This was in part caused by the difficulty in identifying the requirements of such a language. The incompleteness of the meta-language made it difficult to create comprehensive formal definitions for many of the concepts presented in this paper. Also, the lack of a complete meta-language made proofs impossible, and proofs are necessary for future research and applications. This is a shortcoming that would require a more thorough consideration of the texts available on the subject.

In addition to not providing a high level of precision, this paper was not particularly comprehensive when covering the algorithmic attributes of PPESs. A precise set of attributes was not reached in the process of researching this topic. The precise nature of the algorithmic attributes would require a more rigorous and formal approach than the one that was presented in this paper. In particular, a more complete set of constraints was not identified. A more comprehensive list of possible categories of PPESs was also not identified. This is one of the most important aspects of this line of research.

Without a firm understanding of the relationship between algorithmic attributes and output attributes, inferential statistical models were difficult to create. This was compounded by the lack of knowledge related to random equation generation. Without a known way to create random equations with properly constrained numerical attributes, the statistical models that are

produced are not accurate. For this reason, the statistical models created in this paper were simplistic and not analyzed in depth. The primary reason for including these models was to demonstrate the avenues of research that are possible if an effective method of random equation generation is created.

5.3 Future Research

Since this paper was only intended to begin to conceptualize the areas of PPEs, there are numerous possible topics that can be researched in order to improve the understanding of PPEs. Of particular importance for future research will be a complete definition of a meta-language of algebra along with formal definitions and proofs. This would provide a common language that other researchers could use to build more precise and elaborate concepts and theories. With a solid base provided by a focused meta-language, future researchers could then define more precise algorithmic attributes of PPEs. A knowledge of algorithmic attributes, combined with a better understanding of PPE output attributes, would likely allow the creation of better methods for inferring a PPE's algorithm from its outputs. This in turn, along with student modeling techniques, could hopefully be leveraged to create ITSs.

The improvement of the formal meta-language that is used to describe PPEs is necessary to make progress in the research of PPEs. This includes establishing a more rigorous standard for the meta-language and defining rules of formation for the object-language using the meta-language. With this improved language, better definitions can be made. Once the meta-language is

better formalized, proper proofs can be made about PPEs.

After a more formal system for describing PPEs has been defined, research into categorizing the different types of PPE algorithms can be conducted. This will then allow research into practical applications. This research could leverage the research that has been done on rewrite rule systems to identify the possible constraints that all PPEs must follow. Research into the meta-level effects of traditional strategies for solving equations may also provide more insights into the constraints and possibilities of PPEs. These traditional strategies include collecting like terms, obtaining common denominators and factoring polynomials. Analyzing the outputs of PPE software, even if it is closed source, or interviewing mathematicians may also provide insights into the strategies that can be used to solve equations by challenging the researchers to use describe these strategies using the meta-language.

In addition to researching the algorithmic attributes of PPEs, the nature of the output attributes can also be researched in more depth. It is possible that the research into the algorithmic attributes of PPEs could inspire the creation of more meta-level functions and predicates at the first four tiers of output structures. Of particular importance to practical applications would be the continued research into methods of creating useful multi-solution analysis algorithms. This could initially take the form of improved statistical methods, in particular the creation of adequate random equation generation methods. More insightful statistical models could also be pursued. Further research could also be carried out to use pattern recognition to identify strategies that are being

used by a PPES. A long term goal of research may be to create a system that can use the properties discovered about algorithmic attributes to efficiently query a PPES to infer its algorithmic attributes. It can be seen that there are many avenues available for future research which would likely lead to practical, perhaps even revolutionary, improvements to the teaching of equation solving.

6 Appendices

6.1 *Appendix A: Meta-Level Languages and Object-Level Languages*

In order to understand PRESS, Presston and the conceptual framework that is proposed in this paper, an understanding of the distinction and relationship between a meta-level language and its corresponding object-level language is required. While this knowledge is not particularly complicated, it is not well known outside of certain specialized fields of study. As will be shown, the use of meta-level language/object-level language pairs are present in both natural languages, such as English, and formal languages, such as mathematics. However, these occurrences are almost always informal. This is adequate for most purposes but must be formalized for use in applications like PRESS, Presston and the conceptual framework in this paper.

6.1.1 Formal Languages

In order to make precise statements about some specific area of interest, specialized languages, called formal languages, have been invented. These languages sacrifice the flexibility of natural languages, such as English or French, for the precision of the languages found in logic, mathematics and programming. A formal language contains a set of symbols such as "5" , "+" and "=" that may or may not have meanings associated with them. By combining these symbols together into "strings" of symbols, expressions can be made. Formal languages have precise rules for what is considered an acceptable expression in the language and what is not. The rules that state what is an

expression in a language are called formation rules, and expressions that follow these rules are called well-formed formula (WFF, pronounced “woof”) [Sowa 2000 p.470]. For example $1+1=2$ is a WFF of algebra while $1++=2=3$ is not.

6.1.2 Languages to Talk About Languages

A formal language that most people are familiar with is the language of algebra. This language is designed to make precise statements about real numbers. The symbols used in algebra refer to numbers, operations on those numbers and relationships between those numbers (note that this is a simplified coverage of this language). The expressions in this formal language allows unambiguous statements to be made about numbers.

However, when actually doing mathematics, we may wish to make statements *about* the expressions that are being dealt with in algebra, but algebra *only* makes statements about numbers. For example, we cannot say that the expression $(4+2)+5$ is a sum with the language of algebra. The property of being a sum cannot be applied to a number, and $(4+2)+5$ is a number, namely the number 11. We cannot say that $(4+2)+5$ is a sum in the language of algebra without conceding that the number 11 is also a sum, since they are equal. We are thus making a distinction between the arrangement of symbols used in in expression, usually called the expression's syntax, and the meaning of that expression, usually called the expression's semantics.

6.1.3 Meta-Languages

Usually we would just use English, or some other natural language, to make statements about algebra. However, if we wish to say something more precise and complex about a given formal language, a second formal language is invented. The formal language that is being talked about is called the "object-language". The formal language that talks about the object-language is called the "meta-language". The meta-language therefore is above and dependent on the object-language. It does not make sense to talk about a meta-language without having an object-language.

Terms such as "sum", "polynomial", "equation", "solved equation", "coefficient" and so on are part of the meta-language of algebra. The following terms are necessary to understand the use of meta-level statements made in this paper.

- Object-Level Domain of Discourse
- Object-Level Constants and Variables
- Meta-Level Domain of Discourse
- Meta-Level Constants and Variables
- Object-Level Predicates and Relations
- Meta-Level Predicates and Relations
- Expressions

6.1.3.1 *Object-Level Domain of Discourse*

The domain of discourse, also known as the universal set, for the object-level of algebra is the set of real-numbers. This will be denoted \mathbb{R} .

6.1.3.2 Object-Level Sentences

The object-Level sentences of algebra consist of an extension of second-order logic and thus may contain logical symbols such as " \neg ", " \wedge ", " \vee ", " \Rightarrow ", " \Leftrightarrow ", " \forall " and " \exists ". Relational symbols are also in the object language such as " \in " and " $=$ ". It also contains the numerals "1", "2", "3", "4", "5", "6", "7", "8", "9" and "0". The operator symbols are also included such as "+", "-", "\cdot", "ln", "sin", "cos", etc. For this paper, all letters that are lower case will be variable symbols and constant symbols of the object language. These symbols together will constitute the set of symbols for the object-language. This set, which is at the meta-level, will be called L_0 .

6.1.3.3 Meta-Level Domain of Discourse

The meta-level domain of discourse, or the meta-level universal set, consists of *both* the symbols from the object language, L_0 , and the set of real numbers, \mathbb{R} . This set is therefore $L_0 \cup \mathbb{R}$. When referring to the symbols or arrangement of symbols into expressions, double quotes will be used.

6.1.3.4 Meta-Level Sentences

Sentences of meta-level algebra can express everything that can be expressed using the object language, with the exception that constants, variables and functions of the meta-level appear in uppercase in this paper. However, the meta-level can also make statements about the object-level's sentences and parts of those sentences. When using the meta-level statements about the object-level, the object-level sentence or expression will appear in quotes. Therefore,

saying that ‘ “ $(1+x)+4$ ” is a sum’ is a statement at the meta-level but the statement that $x+1=5\cdot x$ is a statement at the object-level. The first is a statement about an expression and the second is a statement about the number x .

Two object-level expressions are equal at the meta-level if they have the same symbols in the same order. Therefore, $"1+x" = "1+x"$ but $"1+x" \neq "(1+x)"$. A meta-level operator called concat denoted ' \oplus ' takes two strings of symbols and outputs a single string of symbols that is the consecutive combination of the original symbols. Concat is associative but NOT commutative. For example, $"(x+3)" \oplus "+5" = "(x+3)+5"$ and $"5" \oplus "(x+3)" = "5(x+3)"$. The concat operator will often be omitted for brevity along with meta-level parentheses. This means that sentence such as ‘ $(X \oplus "=") \oplus S$ is an equation’ will normally be written as ‘ $X="S$ is an equation’ since they are easier to read. The concat operator has been introduced simply to allow the reader to realize that there is an operation being performed, even if it is not explicitly being written. Note this convention for the concat operator is much like the convention used for omitting multiplication signs such as in $5x^2$. This paper will not always be highly formal and symbolic, usually resorting to English, in order to maintain the ease of readability.

Single symbols from the object language of algebra can be placed into sets:

1. The set of numerals: $NUMERALS = \{"1", "2", "3", "4", "5", "6", "7", "8", "9", "0"\}$
2. The set of operator symbols: $OPERATORS = \{"+", "-", "\cdot", "\div", "\wedge"\}$

3. The set of relation symbols: $RELATIONS = \{ "\wedge", "\vee", "\neg", "\Rightarrow", "\Leftrightarrow" \}$

An important aspect of expressions, which will form the foundation of WFFs and expression trees, is the concept of an “atomic expression”. An atomic expression is one or more consecutive symbols, possibly in a larger expression, meant to be considered as part of a whole. For example, the expression "sin" contains three symbols, but is meant to be read as a single expression. The following consecutive symbols are atoms:

1. Any consecutive collection of symbols that contain numerals and at most one decimal point is an atom.
2. Any consecutive collection of symbols that contain lower case letters is an atom.
3. All of the symbols in the single symbol sets are atoms.

Several predicates and corresponding sets are introduced:

1. The *NUMBER* predicate returns true if a string of symbols that is a number atom is inputted.
2. The *OPERATOR* predicate returns true when an operator symbol is inputted.
3. The *RELATION* predicate returns true when a relation symbol is inputted.

6.2 Appendix B: The Language of Expression Trees

The conceptual framework in this paper makes heavy use of the language of expression trees. The language of expression trees is an extension of the meta-

language using graph theory that allows the visualization of the structure inherent to all well formed expressions of the object language. More importantly, it contains concepts that are essential for understanding PRESS, Presston and all other PPEs. The language of expression trees explicitly shows the composition of functions and operations along with the constants and variables that are the first inputs to these functions and operations. This is done by using tree graphs from graph theory. These graphs are technically directional. However as will be shown, the directional nature of these arcs is omitted and is instead indicated using vertical “levels”. Note that the equals sign, along with all other predicates and relations, is considered an operation in expression trees.

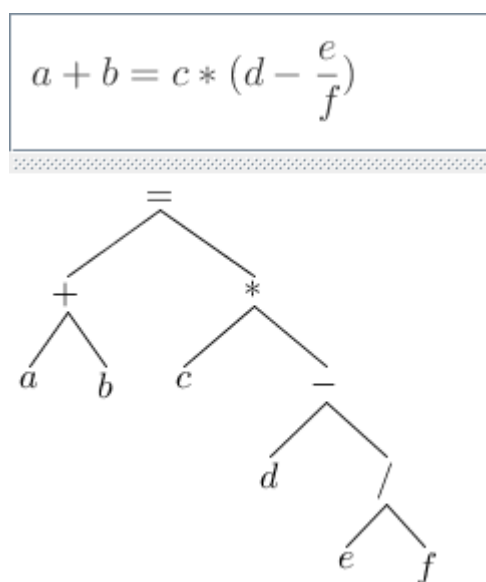


Figure 17: Expression Tree

Expression trees consist of “nodes” connected by “arcs”. Each node is a symbol representing a function, operation, constant or variable of the object language. Note that these symbols are constants of the meta language. In the

following diagram, all of the tree's nodes are highlighted.

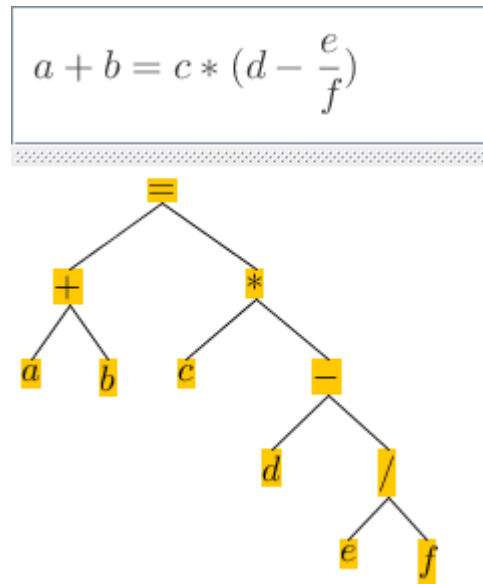


Figure 18: Nodes

The arcs indicate which functions, operations, constants and variables are inputs to a given function or operation. As will be shown, the vertical and horizontal positioning of the symbols indicate the interpretation of each arc between two symbols. In the following diagram, all of the tree's arcs are highlighted.

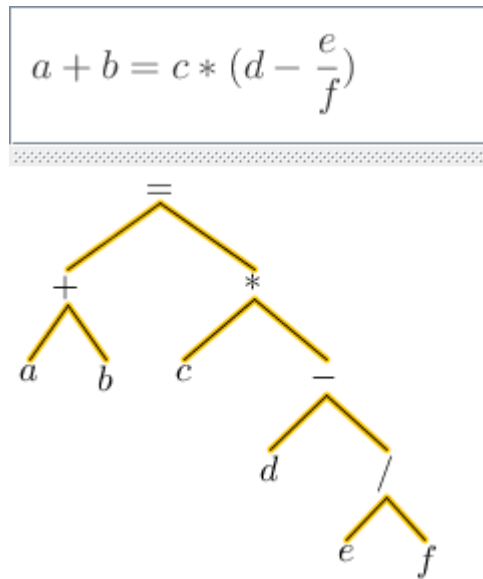


Figure 19: Arcs

The node at the top of a tree is called its “root” node.

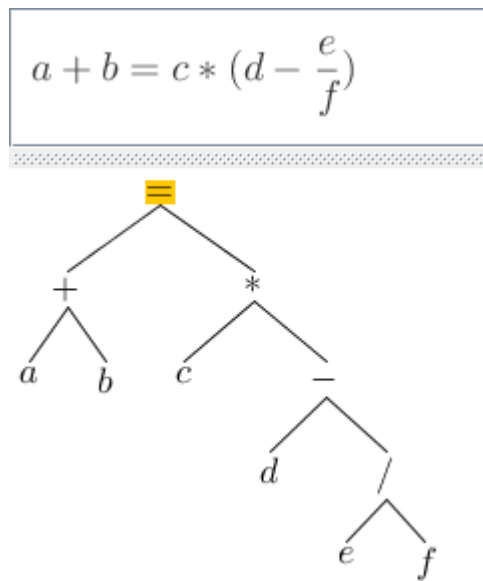


Figure 20: Root Node

The nodes that are connected immediately below a given node by arcs are its “child” nodes. The “children” of a node are the inputs to that node. The left-most child of a node is that node’s first input, the next child to the right is its second

input and so on. In the following diagram, the root node has two children, and it is the “parent” node of these children. Children nodes of the same parent node are called “sibling” nodes.

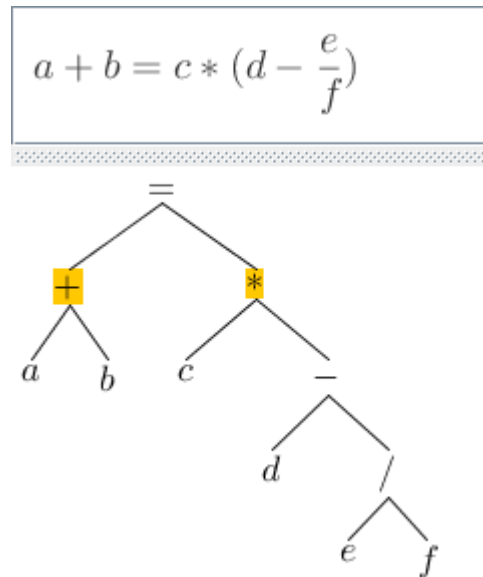


Figure 21: Child Nodes

Nodes that don't have any children are called “leaf” nodes. Leaf nodes will always be variables or constants of the object language because they do not have inputs.

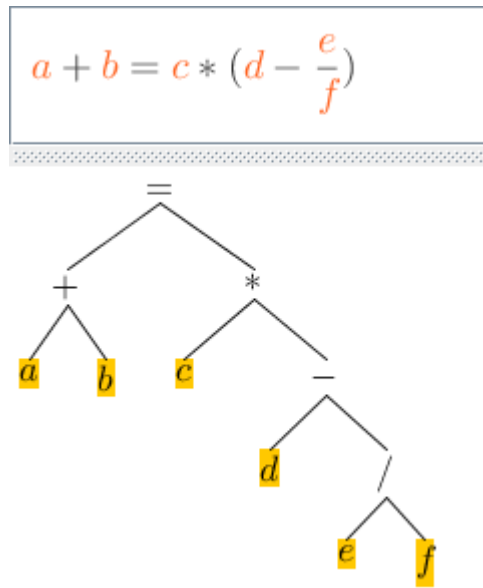


Figure 22: Leaf Nodes

Each node below the root node in a tree is located at a specific “position” in the tree which is determined by the arcs that need to be followed from the root node to arrive at it. Each arc that connects a parent node with its children is labeled with a number. The leftmost child is labeled 1, the sibling to its immediate right is labeled 2, and so on. The following diagram shows the tree with all of its positions indicated.

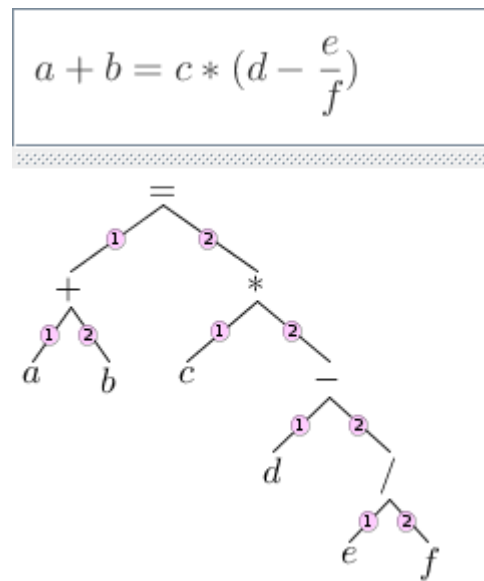


Figure 23: Node Positions

The children, children of children, etc. of a given node are called its “decedents”. A node along with all of its descendants and the arcs between them form a “subtree”. The topmost node in a subtree is the “dominate” node, and all of its descendants and arcs between them are said to be “dominated” by the dominant node. In the following diagram, the subtree that has the node “-” as its dominant node is highlighted.

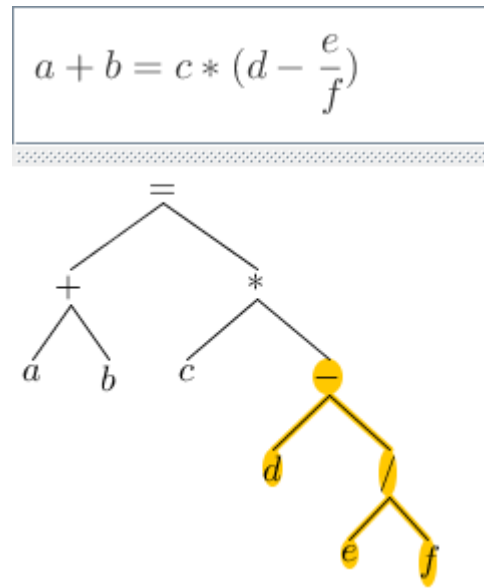


Figure 24: Dominant Node

The position of node "b" is [1 2] . A sequence of arcs and nodes is called a "path", and a path that goes from the root node to a leaf node is called a "branch". The following diagram shows the branch from the root node to node "b" .

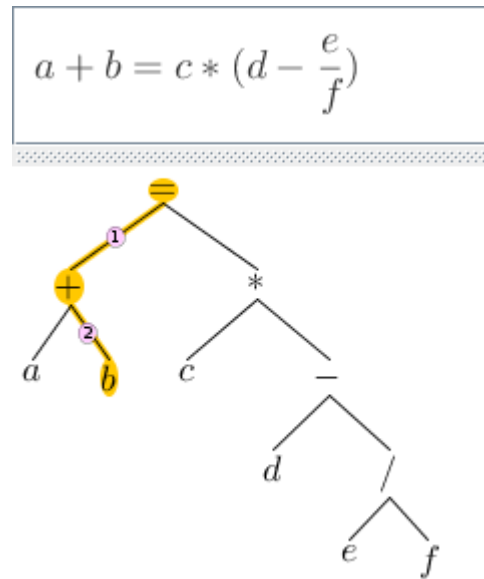


Figure 25: Node Position

The position of node "-" is [2 2] . The following diagram shows the path from the root node to node "-" .

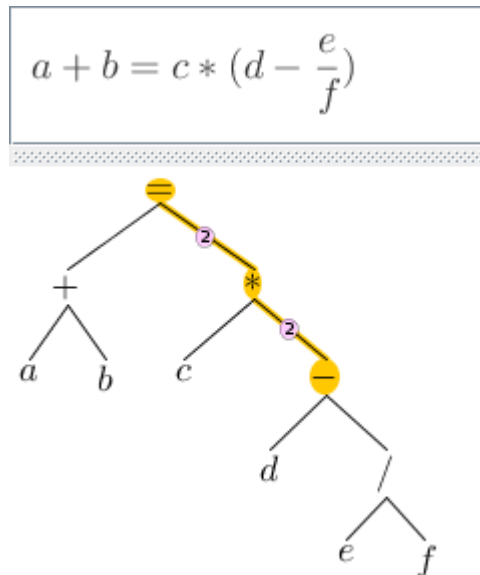


Figure 26: Node Position

The "length" of a path is the number of arcs it contains. For example, the

length of the path between node "a" and node "d" is 5.

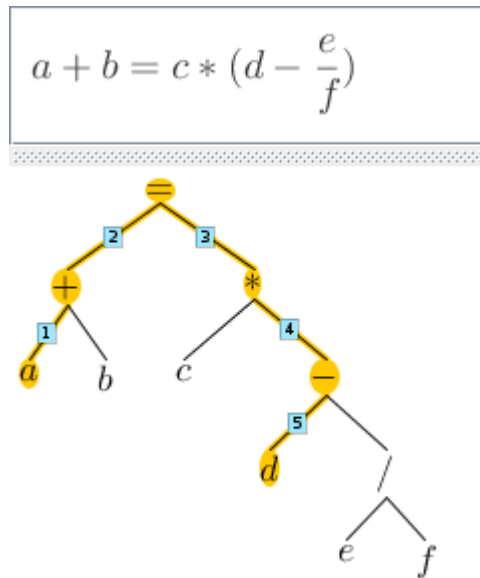


Figure 27: Path Length

The length of a path from the root node to a given node is called the “depth” of the node, and the length of the path from a node to the farthest leaf it dominates is called its “height”. The following diagram shows the depths of all the nodes in the tree.

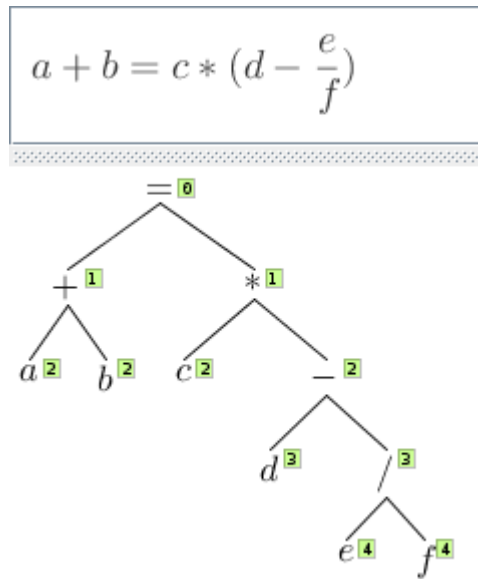


Figure 28: Depths

6.3 Appendix C: Pattern Matching and Rewrite Rules

The patterns used in pattern matching are part of the meta-language, and they are used for creating predicates that can be applied to expression trees. The following are simplified patterns:

- $M_ " + " N_$
- $"1" " \cdot " M_$
- $M_ INTEGER " \cdot " (" N_ " + " K_ ") "$

The variables followed by underscores are called “pattern variables”. A pattern is “used” on a tree, and that tree will either match the pattern or not. The tree matches the pattern if there exists expressions that can be substituted for each of the variables that makes the pattern equal to the expression tree being matched.

For example the expression $"1+3"$ matches the pattern $M_ " + " N_$ because

when $M_$ is replaced by "1" and $N_$ is replaced by "3" the pattern with these substitutions is equal to the expression tree of "1+3". The *INTEGER* meta-level predicate attached to the $M_$ in the pattern

$M_ \text{INTEGER} \cdot " ("N_ + "K_)"$ indicates that the only type of expression that the pattern variable can be matched to is an integer literal. This means that the expression "1·(4+3)" will match to the pattern $M_ \text{INTEGER} \cdot " ("N_ + "K_)"$, but the expression "(1+3)·(4+3)" will not match. Pattern matching consists of taking a pattern and finding all of the locations in a tree that match it.

Rewrite rules are used by meta-level functions to rewrite expressions. Rewrite rules use pattern matching to determine if an expression should be rewritten. The tree is "searched" by the function, and the first position in the tree to match the pattern is rewritten. For example, the rewrite rule

"1" · $M_ \rightarrow M_$ would take the expression "1·(5+2)" and return "5+2".

7 Bibliography

Beeson M (1996), "Design Principles of Mathpert: Software to Support Education in Algebra and Calculus", San Jose State University.

Brown JS and Burton RR (1978), "Diagnostic models for procedural bugs in basic mathematical skills", *Cognitive science*. Vol. 2(2), pp. 155-192. Wiley Online Library.

Bundy A (1975), "Analyzing mathematical proofs (or reading between the lines)", In *Proceedings of the 4th international joint conference on Artificial intelligence-Volume 1.* , pp. 22-28.

Bundy A and Welham B (1981), "Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation", *Artificial Intelligence*. Vol. 16(2), pp. 189-211. Elsevier.

Bundy A (1983), "The computer modelling of mathematical reasoning" Vol. 10

Godoy, G., & Tiwari, A. (2005, July). "Termination of rewrite systems with shallow right-linear, collapsing, and right-ground rules". In *International Conference on Automated Deduction* (pp. 164-176). Springer, Berlin, Heidelberg.

Gries, D., Schneider F. (1995). "A Logical Approach to Discrete Math", Springer.

Nilsson J (1980), "Principles of Artificial Intelligence", Morgan Kaufmann, p.72.

Silver B (1986), "Meta-Level Inference", North-Holland.

Sleeman D (1984), "An attempt to understand students' understanding of basic algebra", *Cognitive Science*. Vol. 8(4), pp. 387-412. Elsevier.

Sowa J (2000), "Knowledge Representation: Logical, Philosophical, and Computational Foundations", Brooks/Cole

Sterling L and Bundy A (1981), "Meta-level Inference in Algebra" Department of Artificial Intelligence, University of Edinburgh.

Sterling L, Bundy A, Byrd L, O'Keefe R and Silver B (1982), "Solving symbolic equations with PRESS", In *European Computer Algebra Conference.* , pp. 109-116.

Van Harmelen F (1991), "Meta-level Inference Systems", Morgan Kaufmann.

VanLehn (1988), "Student Modeling", In M. Polson & J. richardson (Eds.),

"Foundations of Intelligent Tutoring Systems". Hillsdale, NJ: Erlbaum. pp. 55-78.

Wester, M. (1999). A critique of the mathematical abilities of CA systems.
Computer Algebra Systems: A Practical Guide, 16, 436.

<http://MathPiper.org>

8 Biography

Julius A. Kosan

Candidate for the Degree of

Master of Science Mathematical Sciences

Thesis: A Conceptual Framework for Equation Solving Strategies

Major Field: Mathematical Sciences

Biographical:

- Home-schooled prior to collage.
- Started attending Shawnee State University starting in 2010 and graduated with two bachelor degrees and a minor in 2015.
- Started Master's degree in 2017 at Shawnee State University.

Personal Data:

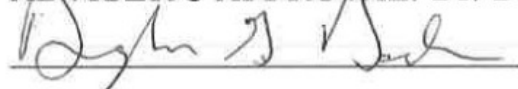
- Email: jakosan@gmail.com

Education:

- Bachelors in Mathematics with a concentration in Statistics
- Bachelors in Natural Science with a concentration in Physics
- Minor in Philosophy

Completed the requirements for the Master of Science in Mathematical Sciences, Portsmouth, Ohio in August 2019.

ADVISER'S APPROVAL: Dr. Douglas Darbro

 8/10/2019