PROBABILISTIC GRAMMAR-BASED EQUATION DISCOVERY

Jure Brence

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Sašo Džeroski, Jožef Stefan Institute, Ljubljana, Slovenia
Co-Supervisor: Prof. Ljupčo Todorovski, Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia

Evaluation Board:
Prof. Bogdan Filipič, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Roger Guimerà, Member, Department of Chemical Engineering, Universitat Rovira i Virgili, Catalonia, Spain
Dr. Jovan Tanevski, Member, Institute for Computational Biomedicine, Heidelberg University, Heidelberg, Germany

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Jure Brence

PROBABILISTIC GRAMMAR-BASED EQUATION DISCOVERY

Doctoral Dissertation

ODKRIVANJE ENAČB Z VERJETNOSTNIMI GRAMATIKAMI

Doktorska disertacija

Supervisor: Prof. Sašo Džeroski

Co-Supervisor: Prof. Ljupčo Todorovski

Ljubljana, Slovenia, April 2024

Acknowledgments

I thank my supervisor Sašo Džeroski for accepting me into his research group and supporting me throughout the journey of becoming a *doctor philosophiae*.

My co-supervisor Ljupčo Todorovski for guiding me on the path of a researcher, for helping me learn an entirely new field, and for our many inspiring and engaging discussions.

The members of the evaluation board: Jovan Tanevski, Bogdan Filipič and Roger Guimerà for the effort they put into carefully reading this work and providing constructive criticism.

My colleagues Nina Omejc, Boštjan Gec and Sebastian Mežnar for our productive collaboration in the field of equation discovery and the work they've done on ProGED, as well as all the laughter we've enjoyed in the office and all the gossip we've shared at lunch.

My colleagues Tomaž Stepišnik, Matej Petković, Blaž Škrlj and Martin Breskvar for their loyal encouragement, their constructive criticism, and for every round of drinks.

The members of the Department of Knowledge Technologies, for helping create an enjoyable and stimulating research environment.

My friends, with whom I regularly shared the triumphs and frustrations of my PhD studies.

Most of all, I thank my family for all their encouragement, especially my mother Ivanka Bračun for her unwavering support and pride in each and every one of my endeavors, my uncle Drago Bračun for setting me on this path, and my fiancée Petra Mikolič for her love and for our many discussions on statistics. But mostly the love, of course.

Finally, I acknowledge the financial support from the Slovenian Research and Innovation Agency (ARIS), primarily through the research project N2-0128 and the core funding P2-0103.

Abstract

In this thesis, we introduce novel methods for equation discovery (ED), based on the use of probabilistic grammars. ED and symbolic regression address the task of finding a symbolic mathematical model that best describes observed data. Models can be as simple as an algebraic equation or as complex as a system of differential equations. Traditionally, domain experts derive equations based on theory and use regression methods to estimate their parameters. ED methods seek to automate the identification of equation structure as well as its parameters. The advantage of discovering closed-form equations over black-box models, popular in machine learning, lies in their inherent interpretability and correspondence with domain theory.

Our methods focus on the use of probabilistic context-free grammars (PCFGs) as a tool for generating mathematical expressions, constraining the space of expressions and encoding background knowledge. We demonstrate that PCFGs parametrize the parsimony principle inherently and intuitively. Furthermore, in addition to the hard constraints imposed by CFG, PCFGs allow us to impose soft constraints on the search space of mathematical expressions. To aid analysis, we introduce a novel method for visualizing the search space of expressions, useful for any ED approach. We introduce a Monte-Carlo algorithm that enables the use of PCFGs in ED and perform extensive computational experiments using an established benchmark database. The results demonstrate that our approach can be used to discover equations, but performs worse than existing methods.

To improve the performance of ED, we introduce dimensional attribute grammars, an extension of PCFGs, that generate only dimensionally consistent mathematical expressions. Our computational experiments demonstrate the impact of dimensional consistency in ED, resulting in performance, comparable to state-of-the-art approaches in the field.

We further extend the ideas of attribute grammars into a general-purpose framework for encoding background knowledge. The framework relies on probabilistic attribute grammars to overcome the limitations of PCFGs in expressing complex types of background knowledge. We demonstrate the utility of the framework by designing and analyzing grammars that encode three different types of background knowledge: dimensional consistency, systems of differential equations for chemical kinetics, and systems of differential equations describing electronic circuits.

Finally, we pave the way toward more intricate PCFG-based ED algorithms by developing a novel Bayesian algorithm for sampling mathematical expressions from a PCFG. The algorithm iteratively updates grammar probabilities to improve the performance of ED and enable the estimation of the posterior distribution. An illustrative computational experiment shows that the algorithm works according to our expectations and improves the performance of ED by guiding the search towards more promising areas of the space of mathematical expressions.

Povzetek

V disertaciji predstavljamo nove metode za odkrivanje enačb (ang. equation discovery, ED), ki temeljijo na uporabi verjetnostnih gramatik. ED in simbolna regresija obravnavata problem iskanja simbolnega matematičnega modela, ki najbolje opisuje izmerjene podatke. Modeli so lahko različnih oblik, od preproste algebrajske enačbe do kompleksnega sistema diferencialnih enačb. Tradicionalno znanstveniki enačbe izpeljejo na podlagi teorije, za določanje vrednosti numeričnih parametrov pa uporabijo regresijske metode. Pristopi ED poskušajo avtomatizirati celoten postopek identifikacije strukture enačbe in njenih parametrov. Prednost odkrivanja preprostih enačb v primerjavi z modeli črnih škatel, ki so priljubljeni v strojnem učenju, leži v njihovi naravni interpretabilnosti in skladnosti z domensko teorijo.

Naše metode se osredotočajo na uporabo verjetnostnih kontekstno-neodvisnih gramatik (ang. probabilistic context-free grammar, PCFG) kot orodja za generiranje matematičnih izrazov, omejevanje prostora izrazov in upoštevanje predznanja. Ena od prednosti verjetnostnih gramatik je parametrizacija načela preprostosti na naraven in intuitiven način. Poleg strogih omejitev, ki jih določa CFG, nam PCFG omogoča uvedbo šibkih omejitev v iskalni prostor matematičnih izrazov. Za pomoč pri analizi predstavimo novo metodo za vizualizacijo iskalnega prostora izrazov, ki je uporabna za kateri koli ED pristop. Predstavimo tudi Monte-Carlo algoritem, ki omogoča uporabo PCFG v ED in izvedemo obsežne računske poskuse z uveljavljeno bazo podatkov. Rezultati kažejo, da naš pristop omogoča odkrivanje enačb, vendar je manj učinkovit kot obstoječe metode.

Z namenom izboljšanja učinkovitosti ED kot razširitev PCFG uvedemo dimenzijske atributne gramatike, ki generirajo le dimenzijsko dosledne matematične izraze. Naši računski eksperimenti pokažejo vpliv dimenzijske doslednosti v ED, saj metoda doseže učinkovitost, primerljivo z najboljšimi metodami na področju ED.

Ideje atributnih gramatik razširimo v splošen okvir za kodiranje predznanja v ED. Okvir temelji na verjetnostnih atributnih gramatikah, ki presežejo omejitve PCFG pri izražanju kompleksnega predznanja. Uporabnost okvira pokažemo z razvojem in analizo gramatik, ki kodirajo tri različne vrste predznanja: dimenzijsko doslednost, sisteme diferencialnih enačb za kemijsko kinetiko in sisteme diferencialnih enačb, ki opisujejo elektronska vezja.

Nazadnje utremo pot boljšim algoritmom za ED na podlagi PCFG z razvojem novega Bayesovskega algoritma za vzorčenje matematičnih izrazov iz PCFG. Algoritem iterativno posodablja verjetnosti gramatike, kar izboljša učinkovitost ED in omogoči oceno posteriorne porazdelitve. Ilustrativni računski poskus pokaže, da algoritem deluje v skladu z našimi pričakovanji in izboljša učinkovitost ED, tako da usmerja iskanje v obetavnejše dele prostora matematičnih izrazov.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

JSI	...	Jožef Stefan Institute
IPS	...	International Postgraduate School
ED	...	equation discovery
SR	...	symbolic regression
ML	...	machine learning
AI	...	artificial intelligence
CFG	...	context-free grammar
PCFG	...	probabilistic context-free grammar
PAG	...	probabilistic attribute grammar
RMSE	...	root mean squared error
ReRMSE	...	relative root mean squared error
AET	...	aggregated expression tree
APT	...	aggregated parse tree

Chapter 1

Introduction

Science is a systematic endeavor of building knowledge by gathering evidence and forming explanations about the world around us. However, due to the inherent complexity of the subject, it is often impossible to account for and understand every detail. As a result, scientists create models – simplified abstractions that can explain observations and make predictions under specific assumptions. Standing upon the shoulders of giants, researchers design new models based not only on gathered observations, but on a vast foundation of theory and existing knowledge. Theory is built over long time frames and strives for generality in its explanations. In contrast, individual models are developed and tested faster and are typically more specific and limited in their scope. Models are evaluated based on the quality of their fit to the data, in other words, how well they describe collected observations.

Models often take the form of mathematical equations, which are composed of variables that correspond to physically observable quantities, operators and functions that define the relationships between the variables, as well as constant parameters. Equations have traditionally been developed by scientists based on their domain knowledge, theoretical assumptions, and confirmed models. However, an increasingly common approach is to start with a large amount of data and attempt to develop a model that fits it, which can then be interpreted to provide insight into the underlying system. This process, called data-driven modeling, historically required a lot of manual labor and guesswork from domain experts, but data scientists have been working on automating this process to accelerate the scientific progress.

Recently, advances in computer hardware have led to the proliferation of machine learning in various fields, including finance, industry and science. Machine learning methods tend to produce models quite different to the types of equations, common in science – the models are large and complex, capable of processing various types of data and predicting observations incredibly well, but are much more difficult to understand and interpret. While accurate models that cannot be interpreted are very useful tools for researchers, they are typically not enough to validate theories and create new knowledge. As the goal of scientific research is often driven by a desire for knowledge and understanding rather than accurate predictions only, there is still a need for models that can produce interpretable and meaningful results while capturing the complexity of natural systems. As such, equations remain one of the primary abstractions that provide insights into natural phenomena in many scientific domains.

Most equations feature constant parameters. In data-driven modeling, the values of constant parameters are typically determined by optimizing the model fit to the data in a process called model fitting. Historically, model fitting was a time-consuming task that required significant mathematical and statistical expertise, often performed by hand. The

advent of computers has made this process much more efficient, enabling automated fitting of the values of constant parameters to data. However, today, discovering the structure of an equation is still a significant challenge, typically addressed directly by humans.

One area of machine learning research focused on this problem is known by a variety of names, such as *equation discovery*, *symbolic regression*, or even a *machine scientist* [1]. The goal of this field is to develop algorithms that can autonomously search for mathematical expressions that fit the data, with little to no human intervention. This approach has the potential to uncover previously unknown relationships between variables and can provide new insights into complex systems.

1.1 Equation Discovery

The problem of equation discovery (ED) we are studying can be formulated as follows. What we have available is a dataset of measured numeric data in a table with the columns representing the variables x_i and the rows representing observations. In the simplest version of the problem, we are trying to find an algebraic mathematical equation of the form

$$x_i = f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), \tag{1.1}$$

that can be used to calculate the values of the i-th variable x_i, based on the $n-1$ variables $x_j, j \neq i$. When reproducing the original measurements, the output of the model should match the data as closely as possible.

However, simply reproducing the dataset is not our goal, since with enough parameters we can learn to approximate any function. What we actually want is for the model to generalize well to new, unmeasured data, and have some correspondence to concepts in already-existing knowledge. In machine learning, the generalization abilities of a model are typically verified by testing its performance on data that was held-out during training. On the other hand, a key idea in science is the principle of parsimony, closely related to Occam's razor – "among options that describe observations well, the simplest explanation is most likely to be correct". By favoring equations with fewer degrees of freedom and mathematical operations involved, we improve interpretability, since domain scientists are more likely to be able to relate a simpler model to existing knowledge in their field. Furthermore, given enough degrees of freedom, any model can perfectly fit the given data. Such a model, however, is very unlikely to predict new, unseen data well. Therefore, a preference for simpler equations can improve the generalization ability of discovered equations. Of course, models that are too simple can fail to describe the complexity of real-world problems. Thus, the task of equation discovery seeks to balance the numerical accuracy of prediction with the simplicity of the models produced [1], [2].

1.1.1 Types of equations

The simplest form of a model is an algebraic equation, which is able to describe only a small portion of the problems studied by scientists. The simplest extension is to systems of equations, where the number of the dependent system variables is m. For this problem to be fully determined, a system of m independent equations is required:

$$x_i = f_i(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), \qquad i = 1, ...m. \tag{1.2}$$

Relations between system variables can make it impossible to express the output variables explicitly as in the form (1.1), but may require an implicit function:

$$0 = f(x_1, \ldots, x_n). \tag{1.3}$$

Studies of dynamical systems often require modeling time series data, requiring ordinary differential equations of the form:

$$\frac{dx_i}{dt} = f(t, x_1, \ldots, x_n). \tag{1.4}$$

Further complications might include partial differential equations or stochastic differential equations. Most of the listed types of tasks can be combined, creating for example, systems of implicit ordinary differential equations. Many methods of equation discovery tend to specialize to certain types of problems that they solve exceptionally well.

Equation discovery systems can typically be separated into two distinct, but closely coupled tasks: identifying the structure of the equation estimating the values of its numerical parameters.

1.1.2 Structure identification

The first part of the equation discovery task is concerned with finding the optimal structure of the equation, a problem typically formulated as a search or optimization in the space of all possible equations, or more practically, a part of the space. There are a handful of concepts to discuss when it comes to structure identification [1].

Representation of mathematical expressions. The language of mathematics is highly complex and diverse, consisting of a wide range of symbols and operations that can be combined in numerous ways. Generating equations or performing a search in the space of all possible equations requires a way of encoding symbols and mathematical operations, as well as rules on how to combine them. There are several methods for achieving this, including enumeration [3], formal grammars [4], [5], and symbolic expression trees [6]. Each of these approaches has its own strengths and weaknesses, and the choice of representation can significantly affect the space of equations that we are able to explore.

Constraining the search space. The space of all possible equations is generally infinite. Following the parsimony principle, which states that the simplest solutions are usually the best, different approaches manage the complexity of equations in different ways. Some classes of equation discovery approaches perform an explorative search in the entire space of equations and rely on sparsity [6] or the minimum description length principle [7], [8] to encourage parsimony. Other approaches seek to limit the number of candidate equations using a variety of methods and employ an exhaustive or intensive search strategy in a constrained search space [9].

Representation of background knowledge. In order to constrain the search space in an informed manner, it is important to leverage background knowledge. This knowledge can come in various forms, including universal principles that apply across different domains, such as the theory of dynamical systems [10], or domain-specific beliefs and assumptions held by experts in a particular field [11]. An important question in ED is how to effectively express and leverage different types of background knowledge. Existing approaches include process-entity formalisms [12], formal grammars [5], prior probability distributions [8] and general mathematical knowledge, such as dimensional analysis, symmetries and various heuristics [13].

Interpretability of results. Equation discovery (ED) methods typically generate closed-form equations as their output, which makes them highly interpretable, provided that the resulting equations are also parsimonious. As such, ensuring parsimony is a requirement for many ED methods. Some approaches go beyond that and allow for higher interpretability by providing correspondence to domain knowledge [12], a posterior distribution over the results [8] or a Pareto front of equations with varying complexity [13], [14].

1.1.3 Parameter estimation

Parameter estimation is a critical component of equation discovery, as it involves identifying the optimal values for the constant parameters of an equation. As such, it is a well-studied problem that has been extensively researched and investigated. Typically, the task of parameter estimation is formulated as an error minimization optimization problem, where the objective is to minimize the difference between the predicted and observed values [1], [15].

There are two general classes of minimization algorithms that are commonly used to solve the parameter estimation problem: local and global methods [16]. Local methods are typically based on local gradients and are designed to converge quickly. However, one significant disadvantage of these methods is that they often get trapped in a local minimum, requiring the user to restart the algorithm multiple times with different initial values to obtain a satisfactory solution. Examples of local methods include direct-search methods, such as Nelder-Mead [17], and gradient-based methods, such as gradient descent and Newton's method [18].

In contrast, global methods are slower but are capable of identifying the global minimum. Popular examples of global methods include random search, evolutionary algorithms [19] and ant colony optimization [20]. Although parameter estimation is a crucial task in equation discovery, it is, in essence, a question of numerical optimization. In this thesis, we focus on the task of structure identification, which involves discovering the functional form of the equation itself.

1.2 Existing Work on Equation Discovery

The field of automated equation discovery dates back to 1981, when Langley established the key concepts and developed a pioneering modeling system called BACON [21]. The approach mimics that of a human scientist, where the system analyzes data to identify patterns and regularities, forming hypotheses based on data-driven heuristics. Specifically, BACON examines pairs of system variables to identify trends and constancies, which allows the algorithm to create new synthetic variables treated as data, building a mathematical description of the problem iteratively. For instance, when two quantities are found to have an inverse mutual dependency, the algorithm creates a new variable by multiplying them. BACON has successfully rediscovered the ideal gas law, Kepler's third law of planetary motion, Coulomb's law, Ohm's law, and Galileo's laws for the pendulum and constant acceleration.

BACON inspired a family of modeling systems that adopted its ideas and added their twists and extensions, such as FAHRENHEIT by Koehn and Zytkow in 1985 [22], EF by Zembowitz and Zytkow in 1992, E* by Schaffer in 1993 and Goldhorn by Križman in 1995. These modeling systems were limited to algebraic equations. An interesting approach was taken in the development of ABACUS by Faklenhainer and Milchalski in 1996 [23]. This modeling system relies on heuristic reasoning and introduces a number of new techniques, including proportionality graph search, suspension search, and notably, dimensional analysis, and is also capable of discovering piecewise equations through the use of clustering. A significant contribution of ABACUS is the concept of the generate-and-test methodology, where the search space is searched through by generating candidate equations, which are then evaluated. This concept has become the most prominent approach in equation discovery over time [1].

In 1995, Džeroski and Todorovski extended the equation discovery field towards differential equations with their LAGRANGE system [24]. LAGRANGE introduced first-order derivatives of system variables and was based on multidimensional linear regression.

The system also introduced new terms by multiplication, following ideas from inductive logic programming and machine discovery systems. The discovery of differential equations complicated the parameter estimation step, as a numericalintegration of the differential equation system had to be performed for each evaluation in the process of optimizing the parameters.

Overall, the development of these modeling systems and extensions laid the foundation for the current state of automated equation discovery, providing a basis for future research in the field.

1.2.1 Knowledge-driven equation discovery

The field of equation discovery has shown a clear need for collaboration between domain experts and machine learning models, as the ability to incorporate domain knowledge can have a significant impact on the accuracy of discovered equations. The development of a practical framework for encoding domain knowledge is crucial in this regard. In 1993, Lindsey and others presented the DENDRAL system, which sought to incorporate task-specific knowledge into the equation discovery process [25]. The heuristics developed in DENDRAL proved to be effective in constraining the search space of equations, but hard-coding knowledge into a system can be limiting.

To address this issue, Todorovski and Džeroski reworked LAGRANGE into LAGRAMGE in 1997 [5]. LAGRAMGE leverages the expressive power of context-free grammars to encode domain knowledge declaratively, thereby allowing domain experts to provide constraints on the space of candidate equations without resorting to hard-coding. This approach has led to more accessible and practical frameworks for encoding domain knowledge.

Process-based modeling is another promising approach that has been used to encode domain knowledge. Introduced in 2004, in this concept, a process-entity formalism was used to describe complex systems [12]. This approach focuses on making the encoding of domain knowledge more efficient and accessible for non-computer scientists. ProBMoT is a state-of-the-art tool in this family of methods [9], [26], [27].

1.2.2 Dimensional analysis

ABACUS [23], introduced in Section 1.3.1, uses dimensional analysis among a variety of other techniques. Also referred to as unit analysis, dimensional analysis makes use of the fact that the physical units of all terms in an equation must match. By examining the dimensionality of system variables, one can impose constraints on the available terms and thus reduce the search space. Dimension analysis has long been recognized as a useful component in equation discovery. The work of Kokar in 1986, who developed a system called COPER, relied heavily on this technique [28]. COPER makes extensive use of the Buckingham Π theorem, a powerful formalization of dimensional analysis that allows for the computation of sets of dimensionless parameters from given variables and provides a path towards constructing an equation.

An extension of COPER, called SDS, was reported on by T. Washio and H. Motoda eleven years later [29]. One limitation of COPER was the requirement that the dimension of each system variable be known, which limited its applicability to non-physical domains. SDS relaxes this requirement and needs to know only the scale type for each of the observables to greatly restrict the number of candidate equations.

1.2.3 Genetic programming

So far we discussed approaches that aim to reduce the search space of equations and use different techniques to narrow down the possibilities, such as heuristics, domain knowl-

edge, and dimensional analysis. However, in 2009, a different approach was introduced by Schmidt and Lipson [6] that was based on the concept of genetic programming [30]. Their method was designed to search the unconstrained space of equations, allowing for the discovery of equations that were not necessarily limited to a pre-defined set of terms or dimensions. This method, commonly referred to as genetic symbolic regression, treated candidate equations as individuals in a population that were evolved using genetic operations such as mutation and crossover.

Genetic symbolic regression has been studied extensively since its inception, and many variations of the method have been developed [31]. In 2006, Ryan proposed a different implementation of genetic programming [32], [33] that employed a formal grammar to define the search space and rules for constructing expressions, as opposed to working with expression trees and applying mutations to them. This allowed for greater control over the search space and guaranteed that generated equations were syntactically correct.

1.2.4 Sparse linear regression

The approach taken by Brunton, Proctor and Kutz in 2016 with their program SINDy [3] differs from other methods in the field of equation discovery. Rather than relying on heuristics, domain knowledge, or genetic programming, SINDy employs the traditional machine learning method of sparse linear regression. This approach involves generating additional features by applying selected transformations on the system variables, and then using a sparsity term in the loss function to favor equations with only a small number of terms. This method is particularly well suited for discovering ordinary differential equations, since they are often composed of a low number of terms and contain functions belonging to a rather small set of mathematical functions.

One advantage of the SINDy approach is that it provides a clear and interpretable model for the underlying dynamics of the system, allowing for better understanding and prediction of its behavior. However, the method has disadvantages, particularly when dealing with noisy or high-dimensional data. Recent research has focused on addressing these limitations and extending the SINDy method to a wider range of systems. However, the most important and fundamental limitation of this approach is its reliance on linear regression, which limits the method to equations linear in parameters.

1.2.5 Composite approaches

In 2019, Udrescu and Tegmark published an intriguing work that has made significant contributions to the field of equation discovery [13]. The modeling system, known as AI Feynman, is a composite of various methods that are applied successively, including dimension analysis and brute force search within a highly constrained space. However, the main novelty of their approach lies in their use of a symmetry and separability analysis, which involves the extensive use of neural networks in order to break down complex problems into a series of simpler ones.

1.2.6 Probabilistic approaches

While random number generation and random search are established techniques in genetic programming, most of the methods discussed thus far have been deterministic in nature. However, in 2020, Guimera proposed a novel probabilistic approach to equation discovery that draws upon Bayesian statistics [8]. This approach uses transformations on expression trees to encode the model space and create a graph of candidate equations. The resulting Bayesian machine scientist then explores the search space using Markov chain Monte Carlo,

which allows it to sample the distribution defined by the equation graphs. The Bayes' theorem is then used to update the posterior, which reflects the likelihood of each candidate equation being the correct one.

One of the key benefits of this approach is that it allows domain knowledge to be incorporated through the definition of the prior distribution. In Guimera's work, the prior distribution is defined by pre-training the model on a corpus of 4080 equations, which were mined from Wikipedia articles. This enables the Bayesian machine scientist to incorporate a broad range of knowledge, as well as to learn from the correlations and dependencies between equations in the corpus.

1.2.7 Neural networks

Artificial neural networks (ANNs) have become a prominent family of machine learning methods over the past decades due to their remarkable performance on a wide range of tasks. However, their notoriously opaque and complex nature makes them challenging to interpret, and therefore, they appear unsuitable for the task of equation discovery. Despite this, some researchers have explored the use of ANNs in this domain. One such effort is the development of EQL by Martius and Lampert [34]. EQL uses an ANN with a specialized architecture designed to learn mathematical equations. The network is trained on data samples, and it outputs an equation in symbolic form. Although the equations obtained from EQL are often not physically meaningful or interpretable, the system has demonstrated some promising results in discovering governing equations in various contexts.

1.2.8 Reinforcement learning

Reinforcement learning is a subset of machine learning where an agent learns to make decisions by interacting with an environment. Through an iterative process of trial-and-error, it learns to associate actions with consequences, with the aim of maximizing a reward signal. This approach leverages the concept of exploration and exploitation. Algorithms such as Q-learning and policy gradients find applications in various domains, like autonomous driving and game playing. In recent years, deep reinforcement learning, which uses neural networks to approximate the decision-making functions, has become popular [35].

In 2021, Petersen developed Deep Symbolic Regression [14], which uses a recurrent neural network to generate symbolic expressions and employs reinforcement learning to train the network based on the degree of fit of the generated equations to the data. Importantly, Petersen introduces a modification of the standard policy gradient technique, called risk-seeking policy gradient. This technique is better suited for equation discovery, because it computes rewards based on the best-performing candidate equation, instead of the average average performance of all candidates. Later upgrades to the method utilize rounds of genetic programming between reinforcement learning iterations to further improve performance. In addition, the method can leverage limited background knowledge in the form of priors and constraints, including dimensional consistency [36].

Crochepierre combined the efficiency of reinforcement learning with the ability to express domain knowledge of grammars in 2022 [37]. Their approach uses a context-free grammar to constrain the search space and a partially-observable Markov decision process to select production rules during the derivation of a candidate equation.

1.2.9 Generative approaches

In recent years, there has been considerable interest in the use of deep generative models to generate mathematical expressions. In 2017, Kusner developed a variational autoencoder for general structured expressions that adhere to a context-free grammar [38]. A

grammar for mathematical expressions allows the autoencoder to be used for equation discovery. However, the method struggles to exactly recover expressions, and the generated expressions are not always syntactically valid.

These shortcomings were addressed in 2023 by Mežnar, who introduced a hierarchical variational autoencoder that constrains the output to binary expression trees, which guarantees the syntactic correctness of generated expressions [39]. The method embeds the space of possible expression trees into a real-valued low-dimensional latent space. Expressions are generated by sampling or otherwise exploring the latent space and decoding the representations into symbolic expression trees.

Transformers, a neural network architecture incorporating self-attention mechanisms [40], have shown incredible results in recent years in natural language, audio, and other applications of machine learning on sequences. In 2021, Valipour showed that transformers can be used to generate candidate symbolic expression structures in the system SymbolicGPT [41].

Whereas the paradigm of addressing the structure identification and parameter estimation separately has become the standard for modern equation discovery approaches, Kamienny and others took a different route in 2022 [42]. Using end-to-end transformers, the approach attempts to directly generate a symbolic expression, including the values of numerical constants, based on the input data.

1.3 Challenges in Equation Discovery

Having reviewed the major strains of research in equation discovery, let us consider the challenges in the field. The open issues map well to the key concepts of structure identification we introduced in Section 1.1.2. We differentiate equation discovery approaches based on how they represent mathematical expressions, if and how they constrain the search space, the employed frameworks for encoding domain knowledge and the interpretability of the results.

1.3.1 Representation of mathematical expressions

Enumeration involves explicitly listing all possible equations within a certain range of complexity [3], [13]. Enumeration-based representations are known for their simplicity and predictability, offering an easily controllable search space. The complexity of the resulting equations is directly related to the number of elements combined. However, these representations have limited expressive power, as they rely on the researcher's inclusion of all relevant options. Failure to include certain options may result in the model being overlooked.

Symbolic expression trees offer another way to encode mathematical expressions [6], [8], [14]. In this approach, the equation is represented as a tree structure, with nodes corresponding to mathematical operations and leaves corresponding to input variables or constants. This representation can be particularly useful for visualizing and manipulating equations, as well as for performing symbolic simplification and differentiation. Methods based on modifying expression trees are powerful, but also unpredictable. By mutating these trees, researchers can generate equations in forms that were never considered before. As a result, these trees can produce more intricate and complicated models than combining listed elements ever could. However, this power comes at a cost. The generated models are typically complex and contain pointless terms or nonsensical properties that human researchers would instantly disregard.

Grammar-based approaches provide a more structured approach to equation discovery

[5], [43] and lie somewhere in between enumeration and expression tree transformations in terms of their capabilities. These approaches have a higher expressive power than enumeration, as just a few productions can define a vast number of models. However, the equations generated are still bound by the rules specified by the grammar. As a result, the expressive power of grammar-based methods is lower than that of expression tree transformations. Nevertheless, grammar-based approaches are more reliable, resulting in fewer nonsensical equations, and the complexity of the generated equations is easier to control. Although they may not be as intuitive as listing terms, grammar productions are arguably easier to comprehend than transformations on expression trees.

1.3.2 Constraining the search space

Different approaches to equation discovery control the size and complexity of the search space of equations in different ways. Some methods perform an exhaustive search across the entire space of possible models and rely on sparsity [6], [44] or the minimum description length principle [7], [8] to encourage simpler, more parsimonious equations. These approaches can be computationally intensive and may require extensive tuning of hyperparameters to balance the competing objectives of model accuracy and simplicity.

Other approaches seek to limit the search space of candidate equations using various techniques. For instance, one approach might constrain the search to a specific class of functions or impose restrictions on the types of mathematical operations allowed in the equation [3]. Grammars and process-entity formalism allow for more complex and detailed specifications of a search space [5]. A very constrained search space allows an equation discovery method to employ an exhaustive or intensive search strategy [9], [11]. These methods can reduce the computational burden of equation discovery, but may also miss potentially useful models that fall outside of the specified search space.

The advancements in computational power and data storage have led to the widespread use of computationally intensive techniques in machine learning. However, with greater power come greater demands, and in practical scenarios, the available computation time often becomes a limiting factor. The number of potential models in equation discovery grows exponentially not only with the dimensionality of the problem but also with the number of mathematical operations and functions we wish to allow. If we possess reliable and valuable information about the problem, constraining the search space is a prudent choice that significantly increases our chances of solving the problem in a reasonable amount of time. On the other hand, if we have limited knowledge of our system, an unconstrained search may be less likely to exclude the best models before we even start the search.

1.3.3 Background knowledge representation

The question of how heavily and in what manner to constrain the search space is closely related to the inclusion of domain knowledge in the modeling process. A large number of methods provide no dedicated framework for the expert to leverage their knowledge. However, if the system expresses language through enumeration of terms, the expert has the option of modifying the lists of terms and functions. On the other hand, process-entity formalisms have been proposed to represent knowledge about the structure and behavior of the system being studied [12]. Formal grammars have also been used to capture domain-specific knowledge [5], [43], and prior probability distributions have been employed to encode beliefs about the relationships between different variables [8]. General mathematical knowledge, such as dimensional analysis [43], [45]–[47], symmetries [13], and heuristics [21], has also been utilized to guide the equation discovery process. By incorporating background

knowledge into the search for equations, researchers can constrain the search space and increase the likelihood of finding meaningful and interpretable models.

Automated modeling systems are primarily designed to aid researchers from different fields. As a result, these programs must be user-friendly and easy to use, while also being capable of accurately representing knowledge in a helpful way. When assessing knowledge frameworks for different domains, both the formalism's power and its accessibility to non-experts are crucial factors to consider. Modifying lists of terms and functions ranks low in terms of both accessibility and expressive power. While it is conceptually easy to understand, it necessitates a considerable mathematical background from the user to list every property of their problem in the form of a function or a term. The power of enumeration is also limited.

In situations where domain experts possess knowledge about the mathematical properties of their system, methods that allow for their phenomenological incorporation can perform exceptionally well. For example, specifying measurement units in detail, which is typical in the physical sciences, can considerably enhance the performance of AI Feynman [13], [48]. These frameworks' expressive power is typically low since they rely on a limited number of usable mathematical properties.

The process-based modeling paradigm [12], [26], [27] begins with an accessibility and interpretability perspective. In many scientific fields, such as chemistry and biology, process-entity models are the standard way of describing dynamical systems. Process-based equation discovery provides these experts with an easy way to express their domain knowledge. The power of this framework is usually low, relying on a limited number of operations and functions and necessitating manual entry for more complex concepts like polynomials.

In contrast, formal grammars provide significant expressive power and the ability to describe large model subspaces with minimal input [5], [32], [33], [38]. However, grammars are familiar to computer scientists and linguists. Domain experts with prior experience can readily understand and interpret grammars, but those from other domains may require additional training and study. Additionally, designing or modifying an appropriate grammar for equation discovery necessitates a strong mathematical background, ranking the general accessibility of these approaches as low.

Probabilistic approaches offer another possibility. A domain expert can specify a prior probability distribution over the model space to represent their expectations and knowledge about the system. However, this option remains largely unexplored thus far [8].

The way machine learning operates involves training a model using a dataset and subsequently using the trained model to analyze fresh data. A comparable concept can be employed to deduce domain expertise from a set of fundamental equations that represent the relevant scientific field. This technique has been employed as a pre-training measure for the probabilistic Bayesian machine scientist [8], in which the prior distribution was automatically deduced from a set of 4080 equations mined from data. Although not much other research has been conducted in this direction thus far, it is a promising approach that merits additional exploration.

1.3.4 Interpretability of discovered equations

The interpretability of results is a crucial aspect for practical usability of equation discovery. One aspect of interpretability relates to the incorporation of domain knowledge, where process-based modeling is the most advanced since it links the optimal equation to the entity-process formalism that domain experts can easily understand [12]. Users of grammar-based approaches can study the derivation trees of discovered equations, which allow for correspondence with higher-level concepts in the domain theory [5].

Another kind of interpretability is possible with a probabilistic approach, where expressing results as a posterior distribution of equations with associated uncertainties offers additional information to researchers [8]. Certain approaches can provide scientific insights by discovering symmetries, which go beyond just generating an equation [13].

The requirement of parsimony is crucial for meaningful and interpretable equations, and all discussed approaches limit or penalize the complexity of candidate equations in some way. A transparent way of addressing parsimony that is sometimes employed entails providing the user not with a single equation, but instead generating a Pareto front of equations. This allows the domain expert to choose the trade-off between the complexity and the accuracy of the solutions [13], [14].

1.4 Probabilistic Grammar-Based Equation Discovery

Grammar formalisms are not new to the field of equation discovery, but so far research has focused on deterministic context-free grammars and not their probabilistic counterparts. The topic of this thesis is the research and development of equation discovery methods, based on probabilistic grammars, which we use to:

1. generate candidate equations that conform to the rules of mathematics,
2. impose soft constraints on the search space by specifying rules for how equations are derived and their corresponding probabilities,
3. natively and flexibly parametrize the parsimony principle,
4. express domain knowledge through the structure of the grammar and its parameters.

Candidate equations are generated from grammars using a random sampling algorithm, have their parameters estimated through numerical minimization, and are then evaluated based on their error-of-fit, potentially also taking into account their prior probability.

The novel formalism provides experts a more accessible and powerful way of leveraging domain knowledge. A probabilistic grammar defines a probability distribution over all possible equations, which can be interpreted as imposing soft constraints on the entire search space, whereas existing work either searches an unconstrained model space or imposes hard constraints on it. Soft constraints constitute a more powerful formalism for expressing domain knowledge.

Our approach builds on a limited body of prior work, mainly related to the use construction and analysis of context-free grammars for equation discovery [5], the approach of constraining the search space through the expression of domain-specific knowledge for modelling dynamical systems [12] and the methodology of Bayesian statistics [8].

1.4.1 Purpose

The purpose of this dissertation is to make significant contributions to the fields of equation discovery and symbolic regression by introducing a novel approach based on probabilistic context-free grammars. This approach is designed to address some of the limitations of existing methods by leveraging background knowledge to improve the scope and efficiency of equation discovery. We introduce a novel approach to equation discovery based on probabilistic context-free grammars that overcome the limitations of existing works in the field. This approach allows us to express complex mathematical equations using a formal grammar and to sample from the space of possible equations using probabilistic inference. One of the main obstacles in equation discovery is the size of the search space of equations,

which is generally infinite, as well as its structure, which makes it difficult to search through efficiently. By incorporating background knowledge in the form of constraints, based on both general modeling knowledge and domain-specific knowledge, we can focus the search space and improve the efficiency and accuracy of the search.

In addition to improving the efficiency and accuracy of equation discovery, we also aim to improve the interpretability of the results. Specifically, we consider the parsimony principle, which favors simpler equations over more complex ones and introduces an important bias that helps guide the search in the space of equations. The probabilistic nature of the approach allows us to provide estimates of confidence in the results through the probabilities of the generated equations. By providing interpretable equations and their probabilities, our approach can facilitate scientific discovery and help researchers understand the underlying mechanisms that govern their data.

To achieve our goals, we develop an accessible software tool in the form of an open-source Python library for probabilistic grammar-based equation discovery. This tool allows other researchers to apply our approach to their own data sets and explore the potential of probabilistic context-free grammars for equation discovery.

By pursuing these points, this dissertation aims to provide valuable insights into the potential of probabilistic context-free grammars as a tool for equation discovery and to advance the state-of-the-art in the field of symbolic regression. Ultimately, our purpose is to facilitate scientific discovery by providing interpretable equations that accurately capture the underlying mechanisms of complex systems.

1.4.2 Goals

The goals of the dissertation are to design, implement and evaluate a probabilistic grammar-based approach to equation discovery.

1.4.2.1 Design

To achieve these goals, we have identified several design objectives for our approach to equation discovery. First, we introduce a theoretical groundwork for the use of probabilistic context-free grammars as generators of mathematical expressions. We develop a formalism that allows us to express mathematical equations using a grammar and derive probabilistic inference algorithms that sample from the space of possible equations. This enables us to generate equations that are more likely to be accurate and meaningful.

Second, we design a methodology for analyzing grammars that allows for estimations and predictions of the suitability of specific grammars for equation discovery without requiring intensive computational experiments. We investigate methods for analyzing the structure and complexity of grammars, as well as the corresponding search space, and evaluate their effectiveness in predicting the performance of the grammar in equation discovery tasks. This allows us to design more efficient grammars, which define a space of possible equations that is more limited and contains more promising candidate equations.

Third, we design and demonstrate "dimensionally-aware grammars" that generate expressions with correct physical units. By incorporating domain-specific knowledge about the physical constraints of the system being modeled, we can ensure that the resulting equations are physically meaningful and interpretable. For many problems, such constraints reduce the size of the search space dramatically and thereby improve the efficiency of the equation discovery algorithm.

Fourth, we develop and demonstrate the use of grammars for types of domain-specific modeling knowledge, such as epidemiological models, electronic circuits models, and coupled oscillator systems. These grammars allow us to leverage domain-specific knowledge

to focus the search space and improve the efficiency and accuracy of equation discovery.

Fifth, we integrate and demonstrate the parsimony principle in probabilistic context-free grammars. We investigate methods for favoring simpler equations over more complex ones and evaluate their effectiveness in improving the interpretability and accuracy of the results. This ensures that the generated equations are as simple and understandable as possible, without sacrificing accuracy.

Sixth, we investigate the feasibility of an iterative approach to equation discovery, based on updating the candidate expression generator with the results of testing the candidate equations. This approach allows us to refine the grammar and improve the accuracy and efficiency of the search. By continuously updating the candidate expression generator, we can gradually converge to the best possible solution.

Finally, we investigate the feasibility of formulating an iterative approach to equation discovery as Bayesian optimization and/or inference. This approach allows us to optimize the search process and efficiently explore the space of possible equations. By using Bayesian optimization and/or inference, we can intelligently guide the search towards promising regions of the search space.

1.4.2.2 Implementation

The Implementation section describes the specific steps and methods needed to achieve the goals outlined in the Design section. To do so we pursue the following goals.

Firstly, we develop an algorithm for sampling mathematical expressions from probabilistic-context free grammars. This algorithm is necessary to generate a pool of candidate expressions to be used in the equation discovery process. By sampling from the grammar, a wide range of possible equations can be generated, which increases the chance of finding the best-fitting equation for a given problem.

The second component we develop is an algorithm for equation discovery. This algorithm consists of three main steps: generating candidate mathematical expressions, estimating free parameters, and evaluating the candidates. The algorithm is the core of the equation discovery process and serves to identify the best-fitting equation among the generated expressions. This process enables the identification of mathematical relationships and patterns that may not be immediately apparent from the data alone.

Next, we develop frameworks that allow for the expression of various aspects of background knowledge through the design of probabilistic grammars. These frameworks enable the incorporation of domain-specific knowledge into the equation discovery process, leading to more accurate and relevant results. For example, a framework for epidemiological models may incorporate knowledge about the conservation of the number of individuals, while a framework for electronic circuit models may incorporate knowledge about circuit components and their interactions.

Finally, we develop open-source software for probabilistic grammar-based equation discovery. This software provides an accessible tool for researchers and practitioners to use in their own work and enables the wider adoption and evaluation of the proposed methodology. The software allows users to input their data and background knowledge, and generate candidate equations for analysis. It also provides practical tools for evaluating the goodness-of-fit of equations and estimating confidence in the results.

1.4.2.3 Evaluation

The evaluation of the proposed approach is an essential step in validating its usefulness and effectiveness. This section presents the evaluation goals and the methods used to assess the approach's performance.

The first evaluation goal is to determine the ability of the approach to efficiently express various aspects of background knowledge. One of the advantages of the probabilistic context-free grammar approach is its ability to incorporate domain-specific knowledge into the grammar to guide the discovery of equations. To evaluate the performance of the approach in this regard, we perform case studies that test the approach's ability to express different types of domain-specific knowledge, such as epidemiological models, electronic circuits models, and coupled oscillator systems. The cases are selected to cover different types of modeling problems and are used to assess the approach's ability to express search spaces that match the desired properties.

The second evaluation goal is to assess the ability of the approach to discover the correct mathematical laws, underlying the data. To this end, we use a widely adopted database, containing a variety of equation types, including linearity and nonlinearity in the variables, the parameters, or both. Specifically, we test the ability of the approach to correctly discover the equation that was used to generate the data. We compare the results obtained by the proposed approach with those obtained by established methods for equation discovery and symbolic regression.

The third evaluation goal is to evaluate the computational efficiency of the approach in comparison to established methods for equation discovery and symbolic regression. We track the number of candidate equations that must be evaluated by the proposed approach and compare it with the number required by the established methods. We use an established benchmark to assess the scalability of the approach with respect to the size and complexity of the input data.

Overall, the evaluation goals aim to provide a comprehensive assessment of the proposed approach and its potential to advance the fields of equation discovery and symbolic regression. The results obtained from the evaluation will inform the design of future versions of the approach and guide its application to real-world problems.

1.4.3 Hypotheses

The hypotheses, investigated in this thesis, are the following.

H1 We can design an equation discovery approach, based on probabilistic grammars (PCFGs and PAGs), that overcomes the limitations of existing approaches in ensuring parsimony and expressing different types of background knowledge, and providing a probabilistic interpretation of results.

H2 We can implement the designed approach into a software tool, which enables the discovery of algebraic equations from measured or simulated data.

H3 The developed approach can outperform existing equation discovery approaches in performance, computational efficiency and applicability.

1.4.4 Scientific contributions

Overall, this PhD thesis advances the field of equation discovery by improving the methods for leveraging both general and domain-specific background knowledge.

1. The initial contribution of this thesis is the innovative use of probabilistic context-free grammars in equation discovery. Employing PCFGs to represent background knowledge and generate candidate expressions yields simpler and more accurate equations and enables the use of soft constraints, fundamentally enhancing the equation discovery process.

2. The next contribution involves implementing probabilistic attribute grammars that enable dimensionally-consistent equation discovery. This approach maintains the expressive power of PCFGs, while eliminating physically meaningless equations, which significantly improves the performance and computational efficiency of the equation discovery process.

3. Next, the thesis introduces a general attribute grammar methodology coupled with a novel technique for directly sampling probabilistic attribute grammars. This novel approach is set to capture a wide array of background knowledge, extending the reach of domain knowledge in equation discovery beyond the expressivity of PCFGs.

4. Finally, the development of an algorithm for the iterative Bayesian updating of grammar probabilities overcomes the limitations of simple random sampling and paves the way towards computationally efficient equation discovery. An additional benefit of the Bayesian approach is the approximation of the posterior distribution over mathematical expressions, which improves the interpretability of equation discovery.

1.5 Organization of the Thesis

In this chapter, we introduced the problem of equation discovery and gave an overview of the many different existing approaches. We identified the challenges in the field, discussed how different approaches address the challenges and what are the shortcomings of existing work. In light of the identified challenges, we defined the purposes and goals of this work, the hypotheses to be answered and outlined its contributions to science.

In Chapter 2, we focus on probabilistic context-free grammars as a tool for generating mathematical expressions, constraining the space of expressions and encoding background knowledge. We begin by studying the mathematical properties of grammars and the probability theory underlying probabilistic grammars and compare the utility of deterministic and probabilistic grammars from a theoretical perspective. We also introduce a method for visualizing the search space of expressions, which we use throughout the thesis. We introduce a Monte-Carlo algorithm that enables the use of PCFGs in equation discovery and conclude the chapter by performing extensive computational experiments that demonstrate the use of PCFGs in equation discovery.

In Chapter 3, we address a particular type of background knowledge, common in physical sciences – measurement units. We introduce dimensional attribute grammars, an extension of PCFGs, that generates only dimensionally consistent mathematical expressions. We discuss various challenges of the approach and identify the solutions. Finally, we use computational experiments to demonstrate the impact of dimensional consistency in equation discovery.

In Chapter 4, we extend the ideas of the previous chapter into a general-purpose framework for encoding background knowledge. The framework relies on probabilistic attribute grammars to overcome the limitations of PCFGs in expressing complex types of background knowledge. We demonstrate the utility of the framework by designing and analyzing grammars encoding three different types of background knowledge: dimensional consistency, systems of differential equations for chemical kinetics, and systems of differential equations describing electronic circuits.

In Chapter 5, we focus on algorithmic improvements by developing a Bayesian algorithm that iteratively updates the grammar probabilities to improve the performance of equation discovery and enable the estimation of the posterior distribution. We first introduce the algorithm and its theoretical basis. We then perform an illustrative computational experiment that demonstrates the use of the algorithm and enables insight into

its behavior.

Finally, in Chapter 6, we review and evaluate the performed work in light of the goals and purposes we have set for the thesis. Next, we answer the hypotheses of the thesis and detail its scientific contributions. We conclude the thesis by discussing our suggestions for further work.

Chapter 2

Probabilistic Grammars for Equation Discovery

In this chapter, we introduce the notion of grammars, which originates in computational linguistics. A grammar is used as a formal specification of a language and uses a set of production rules to derive valid strings from the language in question. We argue for the use of context-free grammars [49] – a type of grammar that is powerful enough to specify the language of mathematical expressions. Grammars are typically used to discriminate between strings that are part of a language and strings that are not, a process known as parsing [4]. For use in equation discovery, however, we are interested in applying grammars as generative models. In this chapter, we first formally define context-free grammars and their probabilistic versions, as well as study a few illustrative examples. We proceed by defining a PCFG [50], [51] that encodes mathematical expressions and introduce the task of (probabilistic) grammar-guided equation discovery. Throughout this thesis, whenever the distinction between grammars and their probabilistic counterparts is important, we refer to the former as deterministic grammars.

2.1 Context-Free Grammars

Formally, we define a context-free grammar [4] as the tuple $G = (\mathcal{N}, \mathcal{T}, \mathcal{R}, S)$. The set of terminal symbols $\mathcal{T}$ contains all symbols, appearing in strings that are part of the language. Nonterminal symbols, contained in the set $\mathcal{N}$, do not appear in the language, but are used by the grammar to derive strings and often correspond to more abstract concepts, such as factors or denominators in mathematical expressions. The production rules in the set $\mathcal{R}$ are rewrite rules of the form $A \to \alpha$, where the left-hand side is a nonterminal $A \in \mathcal{N}$ and the right-hand side is a sequence of nonterminals and terminals, $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$. For example, the rule $F \to N/D$ specifies that a fraction F is a sequence of N – the nonterminal symbol for a numerator, $/$ – the terminal symbol for division, and D – the nonterminal symbol for a denominator. To derive a sentence, a grammar begins with a string, composed of a single non-terminal $S \in \mathcal{N}$, and applies production rules to recursively replace nonterminal symbols in the current string with terminals and nonterminals. The final string contains only terminal symbols and belongs to the language, defined by the grammar G.

Consider a simple example of a context-free grammar $G_L = (\mathcal{N}_L, \mathcal{T}_L, \mathcal{R}_L, S_L)$ that derives linear expressions of two variables x and y. The set $\mathcal{R}_L$ includes four production

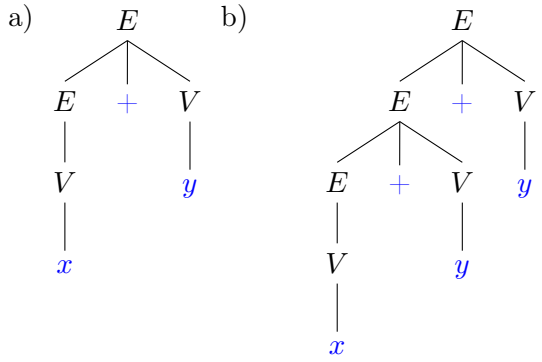


Figure 2.1: Example parse trees for expressions a) $x + y$ and b) $x + y + y$, derived by grammar G_L from Equation (2.1).

rules:

$$\begin{aligned} E &\to E + V \\ E &\to V \\ V &\to x \\ V &\to y, \end{aligned} \tag{2.1}$$

with the set of terminals $\mathcal{T}_L = \{x, y, +\}$, the set of nonterminals $\mathcal{N}_L = \{E, V\}$ and the starting symbol $S_L = E$. The productions of the grammar can generate the sum of an arbitrary number of terms, consisting of the variables x and y in any order. It is one of the simplest grammars for mathematical expressions that demonstrates the concepts we discuss.

The syntactic structure of sentences (mathematical expressions in our case) in a language according to a context-free grammar is represented by parse trees [4]. A rooted, labeled and ordered tree ψ is a parse tree, generated by grammar G, if the root node of ψ is labeled S, and each node in the tree either has no children and its label is a member of $\mathcal{T}$ or there is a production $A \to B$, member of set $\mathcal{R}$, where the label of the node is A and the left-to-right sequence of labels of the node's immediate children is B. The string derived by ψ is the left-to-right sequence of its leaves. The set of all possible parse trees, generated by grammar G is labeled Ψ_G. We provide examples of parse trees ψ_1^L and ψ_2^L for the expressions $x + y$ and $x + y + y$, according to grammar G_L, in Figure 2.1.

The height of a parse tree is defined as the number of edges on the longest path, starting at the root node (starting symbol S) and ending at a leaf (terminal symbol). The parse trees in Figure 2.1 have heights of a) three and b) four. Note that the same mathematical expression derived by a different grammar will in general have a different parse tree and consequently height. Even so, more complex expressions require higher parse trees, making parse tree height an important measure of expression complexity.

2.1.1 Probabilistic context-free grammars

We can transform a context-free grammar into a probabilistic context-free grammar (PCFG) if we assign a probability to each of its productions, so that for each $A \in \mathcal{N}$:

$$\sum_{(A \to \alpha) \in \mathcal{R}} P(A \to \alpha) = 1.$$

In other words, we impose a probability distribution over all production rules with the same nonterminal symbol on the left hand side by ensuring that the probabilities of all

productions with the same nonterminal on the left-hand side sum up to one. Since a given sequence of productions characterizes a single parse tree ψ, the probability of the parse tree is simply the product of the probabilities of all productions that derive it [50]:

$$P(\psi) = \prod_{(A \to \alpha) \in \mathcal{R}} P(A \to \alpha)^{f(A \to \alpha, \psi)}, \tag{2.2}$$

where $f(A \to \alpha, \psi)$ is a function that counts the number of occurrences of the production $A \to \alpha$ in the parse tree ψ. Furthermore, all parse trees derived by a proper PCFG form a probability distribution. In other words, the probabilities of all parse trees sum up to one [51]:

$$\sum_{\psi \in \Psi} P(\psi) = 1. \tag{2.3}$$

Let us again consider the example of the grammar for linear expressions from Equation (2.1). We can transform the CFG to a PCFG by assigning a probability to each of the four productions. We denote the probability of each production in brackets after the production. Note that we also achieve a more compact notation by presenting productions with the same nonterminal on the left-hand side in a single line, separated by a vertical line:

$$\begin{aligned} E &\to E + V \ [p] \mid V \ [1-p] \\ V &\to x \ [q] \mid y \ [1-q]. \end{aligned} \tag{2.4}$$

We introduced parameters $0 < p < 1$ and $0 < q < 1$ to define the probability distributions over production rules for E and V. Following Equation (2.2) we can calculate the probabilities of the two parse trees for expressions $x + y$ and $x + y + y$, depicted in Figure 2.1:

$$\begin{aligned} P(\text{“}x + y\text{”}) &= p(1-p)q(1-q), \\ P(\text{“}x + y + y\text{”}) &= p^2(1-p)q(1-q)^2. \end{aligned} \tag{2.5}$$

2.1.2 Grammars as generators

Grammars were originally developed to formally describe the structure of natural-language sentences, but computer scientists most commonly use them to encode the syntactic structure of markup and programming languages [4]. In this context, grammars enable developers to implement efficient parsers of annotated documents and source code. Given a grammar and a string, the parsing algorithm can determine whether the grammar can derive the string and, if so, finds the corresponding parse tree. The parse tree makes the syntactic structure of the string explicit.

Context-free grammars have been used as generators of mathematical expressions in equation discovery before. LAGRAMGE [5] deterministically and systematically enumerates parse trees from simpler (lower) to more complex (higher) for a given grammar. To do this, the algorithm uses a refinement operator that generates the minimum refinements for a given parse tree by replacing the simpler production rules with more complex ones. The expression generator requires a user-specified maximum height for the generated parse trees. It is important to note that in this procedure, each generated parse tree is equally likely, i.e., it is assumed to be uniformly distributed in the space of parse trees with bounded height.

Probabilistic grammars enable a much more flexible mechanism for generating expressions. Algorithm 1 describes the GENERATE_SAMPLE procedure for randomly sampling the space of parse trees, derived by a grammar. The function receives as input the probabilistic context-free grammar G and the nonterminal root node of the parse tree A, which forms the initial string s.

Algorithm 2.1: GENERATE_SAMPLE(G, A)
Randomly sample an expression from a probabilistic context-free grammar.

Data: Probabilistic grammar $G = (\mathcal{N}, \mathcal{T}, \mathcal{R}, S)$
Result: Expression s corresponding to a randomly sampled parse tree ψ from G with root node A, probability p of ψ

```
1  initialize (s, p) = ([ ], 1);
2  Choose a random rule (A → α) ∈ R : α = A_1 A_2 ... A_k, A_i ∈ N ∪ T;
3  for i = 1, i ≤ k do
4      if A_i ∈ T then
5          s = s.append(A_i);
6      else
7          (s_i, p_i) = GENERATE_SAMPLE(G, A_i);
8          s = s.append(s_i);
9          p = p · p_i;
10     end
11 end
12 return (s, p);
```

By following the list of production rules, the algorithm recursively replaces nonterminal symbols in the string s until s contains only terminal symbols. Whenever more than one production rule applies for a nonterminal A, the function randomly samples a production rule from the probability distribution prescribed by the grammar (line 2). The string s is then expanded using the sequence of symbols on the right-hand side of the chosen production rule. Terminal symbols in the right-hand side are simply appended to the string (line 5), whereas extending nonterminals requires a recursive call (line 7). When the function GENERATE_SAMPLE is called with a grammar and its start symbol, the algorithm generates a randomly sampled string from the grammar, paired with its probability of generation.

Contrasting the deterministic algorithm for generating parse trees from CFGs, used in LAGRAMGE, Algorithm 1 does not require the maximal tree height to be specified. Furthermore, the distribution over the generated strings is in general not uniform, since the probability of each parse tree is the product of the probabilities of all the production rules used in its derivation.

2.1.3 The number of parse trees with limited height

In this section, we compare the properties of expression generators, based on deterministic sampling of CFGs (i.e., LAGRAMGE), with those based on probabilistic sampling of PCFGs. Given a context-free grammar $G = (\mathcal{N}, \mathcal{T}, \mathcal{R}, S)$, we can count the number of parse trees $n_G(A, h)$ with the root symbol $A \in \mathcal{N} \cup \mathcal{T}$ and a height of exactly h using the recursive formula:

$$n_G(A,h) = \begin{cases} 1 & \text{if } A \in \mathcal{T},\ h = 0 \\ 0 & \text{if } A \in \mathcal{T},\ h > 0 \\ 0 & \text{if } A \in \mathcal{N},\ h = 0 \\ \text{number of productions: } A \to w,\ w \in \mathcal{T}^* & \text{if } A \in \mathcal{N},\ h = 1 \\ \displaystyle\sum_{(A\to\alpha)\in\mathcal{R}} \left(\prod_{i=1}^{k} N_G(A_i, h-1) - \prod_{i=1}^{k} N_G(A_i, h-2) \right) & \text{if } A \in \mathcal{N},\ h > 0. \end{cases} \tag{2.6}$$

Here, $\alpha = A_1 A_2 \dots A_k$ represents a string of terminal and nonterminal symbols $A_i \in \mathcal{N} \cup \mathcal{T}$ and $N_G(A, h)$ is the number of parse trees of G with A as the root node and of height up to and including h:

$$N_G(A, h) = \sum_{h_i=0}^{h} n_G(A, h_i). \tag{2.7}$$

Using the above formula with $A = S$ allows us to obtain the total number of parse trees derived by grammar G. In this thesis, we use the shorthand $n_G(h) = n_G(A = S, h)$ whenever the root symbol is not explicitly specified.

Let us demonstrate the use of the above equations by finding the number of parse trees with a given height, derived by the linear grammar in Equation (2.1):

$$\begin{aligned} E &\to E + V \ \mid V \\ V &\to x \ \mid y. \end{aligned}$$

First, notice that there are only two trees with the root V and a height of one:

$$n_{G_L}(V, h) = \begin{cases} n_V & \text{if } h = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

Therefore, $N_{G_L}(V, h) = n_V$ for all $h \geq 1$. We can make use of this observation, considering Equation (2.6) for $h \geq 2$ and the root node E:

$$n_{G_L}(E, h) = n_V - n_V + n_V \left(N_{G_L}(E, h-1) - N_{G_L}(E, h-2)\right) \tag{2.9}$$

$$n_{G_L}(E, h) = n_V \left(N_{G_L}(E, h-1) - N_{G_L}(E, h-2)\right) \tag{2.10}$$

$$n_{G_L}(E, h) = n_V \ n_{G_L}(E, h-1). \tag{2.11}$$

Recognizing this relation as the recursive formula of a geometric sequence, we can rewrite it in the general form as

$$n_{G_L}(E, h) = \begin{cases} n_V^{h-1} & \text{if } h \geq 2 \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

As per Equation (2.7), we can compute the total number of parse trees with height up to and including h as the sum of the first h terms of the geometric series:

$$N_{G_L}(E, h) = \sum_{h_i=2}^{h} n_{G_L}(E, h_i) = \sum_{h_i=2}^{h} n_V^{h_i} = \frac{n_v^h - 1}{n_v - 1} - 1. \tag{2.13}$$

In the example in Equation (2.4) we use a G_L with two variables, hence $n_V = 2$. In this case, the number of parse trees with height up to and including h simplifies to $N_{G_L}(E, h) = 2^h - 2$.

2.1.4 Parse tree probabilities and grammar coverage

In Equation (2.6), we introduced a method for counting the number of parse trees with a given height derived by a context-free grammar. Extending a context-free grammar to a probabilistic context-free grammar does not change the number of parse trees $N_G(A = S, h)$. In order to better characterize the properties of a PCFG, we introduce *coverage*, defined as the sum of probabilities (Equation (2.2)) of all parse trees with height up to and including h:

$$\mathrm{Cov}_G(A, h) = \sum_{h_i=0}^{h} \sum_{\psi \in \Psi_{A,h_i}} P(\psi), \tag{2.14}$$

where $\Psi_{A,h_i} \subseteq \Psi$ represents the set of all parse trees with height h_i a root symbol of A. We can now express Equation (2.3) in terms of coverage as

$$\lim_{h\to\infty} \mathrm{Cov}_G(S,h) = 1. \tag{2.15}$$

Similarly to counting the number of parse trees in Equation (2.6), we can calculate the coverage of a probabilistic context-free grammar at height h using a set of recursive equations:

$$\mathrm{Cov}_G(A,h) = \begin{cases} 1 & \text{if } A \in \mathcal{T},\ h \geq 0 \\ 0 & \text{if } A \in \mathcal{N},\ h = 0 \\ \sum_{(A\to\alpha)\in\mathcal{R}} P(A\to\alpha) \prod_{i=1}^{k} \mathrm{Cov}_G(A_i, h-1) & \text{if } A \in \mathcal{N},\ h > 0. \end{cases} \tag{2.16}$$

Here, $\alpha = A_1 A_2 \ldots A_k$ is the string of symbols $A_i \in \mathcal{N} \cup \mathcal{T}$, appearing on the right-hand side of the production rule $A \to \alpha$.

We can use coverage to demonstrate the benefit of introducing probabilistic grammars by reconsidering the simple linear grammar G_L from Equation (2.1). The grammar contains two probability distributions over production rules we have to specify to fully define the PCFG. The first distribution is over the production rules with E on the left-hand side and the second distribution is over the production rules with V on the left-hand side. Let us assume that all variables are equally probable, which leads to a uniform distribution of the production rules $V \to v$. In other words, $P(V \to v) = 1/n_V$, where n_V is the number of variables. The other probability distribution we must define is over the two recursive production rules $E \to E + V$ and $E \to V$. If we parameterize the probability of the first rule with p, the probability of the other rule is $1-p$. In short, we are studying the following PCFG:

$$\begin{aligned} E &\to E + V\ [p] \ \mid V\ [1-p] \\ V &\to x_1\ [1/p_m] \ \mid \ \ldots\ x_m\ [1/p_m]. \end{aligned}$$

Since this PCFG is very simple, we can derive its coverage with basic probability theory. The probability of generating a parse tree with a height of exactly h is

$$P_{G_L}(E,h) = p^{h-2}(1-p).$$

We can easily see this by considering the Bernoulli process. Note that one of the productions with E on the left-hand side represents recursion, with an associated probability p, and the other terminates the process of generation, with an associated probability $1-p$. In order to produce a parse tree with height h, the generator must choose the recursive option $h-2$ times, then choose the terminal production and finally choose either of the terminal productions for V. With this understanding, we can compute the coverage as the sum of the probabilities of all parse trees with height of up to and including h:

$$\mathrm{Cov}_{G_L}(E,h) = \sum_{h_i=2}^{h} p^{h_i-2}(1-p) = (1-p)\sum_{h_i=0}^{h-2} p^{h_i} = \frac{1-p^{h-1}}{1-p}(1-p) = 1-p^{h-1}.$$

We now demonstrate the usage of the more general Equation (2.16) by using it to reproduce the above result. First, consider coverage for parse trees with V as their root node:

$$\mathrm{Cov}_{G_L}(V,h) = 1,\ \text{if } h \geq 1.$$

Next we derive a recursive expression for the coverage of parse trees with E as the root node:

$$\mathrm{Cov}_{G_L}(E,h) = p \cdot \mathrm{Cov}_{G_L}(E,h-1) \cdot \mathrm{Cov}_{G_L}(V,h-1) \tag{2.17}$$

$$+ (1-p) \cdot \mathrm{Cov}_{G_L}(V,h-1) \tag{2.18}$$

$$\mathrm{Cov}_{G_L}(E,h) = p \cdot \mathrm{Cov}_{G_L}(E,h-1) + (1-p). \tag{2.19}$$

This time guessing the general expression for the h-th term in the series is not trivial. Instead, let us observe the first three terms:

$$\text{Cov}_{G_L}(E, 1) = 0, \tag{2.20}$$
$$\text{Cov}_{G_L}(E, 2) = 1 - p, \tag{2.21}$$
$$\text{Cov}_{G_L}(E, 3) = p(1 - p) + (1 - p) = 1 - p^2. \tag{2.22}$$

From this sequence we can guess the general form $\text{Cov}_{G_L}(E, h) = 1 - p^{h-1}$. To prove it by induction, we assume the relation holds for h and use the recursive relation to prove it for $h + 1$:

$$\text{Cov}_{G_L}(E, h + 1) = p \cdot \text{Cov}_{G_L}(E, h) + 1 - p = p(1 - p^{h-1}) + 1 - p = 1 - p^h.$$

The coverage of the linear grammar is therefore:

$$\text{Cov}_{G_L}(E, h) = 1 - p^{h-1}, \quad \forall h \geq 2. \tag{2.23}$$

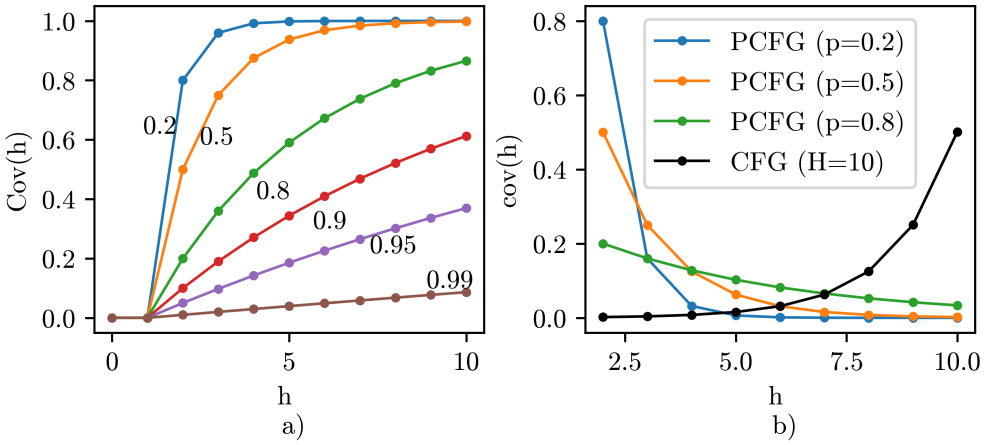


Figure 2.2: Parsimony in context-free grammars: a) the coverage of the probabilistic grammar for linear expressions at a given height h for different values of p – the probability of the recursive rule $E \to E + V$, b) the probability of generating a parse tree with a given height h using the probabilistic grammar with different values of p (colors) and using the deterministic version of the grammar (black line).

This result reveals the impact that p (the probability of the recursive production rule $E \to E + V$ in the linear grammar G_L) has on the probability of sampling a parse tree with a given height. In the left-hand side of Figure 2.2 we depict the total probability of all parse trees with height up to and including h (i.e., the coverage) for different values of p. The simplest parse trees encoded by grammar G_L correspond to expressions x and y, with height 2 and a probability of $1 - p$. The probability of sampling of the simplest trees is high for small values of p. On the other hand, the likelihood of sampling higher parse trees, corresponding to more complex expressions, increases with p. By varying p, we can directly control the degree of complexity in generated equations, which represents an intuitive parametrization of the parsimony principle in equation discovery. In contrast to regularization constants in sparse regression or genetic algorithms, which typically have unlimited range and arbitrary meaning, the probability of recursion in PCFGs has a straightforward probabilistic interpretation.

The parametrization of the parsimony principle when using PCFGs as generators of expressions is also more elegant when compared to the use of deterministic grammars. Since we assumed a uniform distribution over the production rules for V in the PCFG G_L, the probability distribution of the parse trees with a given height is also uniform. The likelihood of sampling a parse tree with a certain height h decreases exponentially with h, due to the fact that the number of parse trees with height h increases exponentially with h. However, in deterministic grammars, the sampling probability is distributed uniformly across all trees, while the number of parse trees still increases exponentially with h. Consequently, the probability distribution over tree height in deterministic grammars is biased towards more complex trees – the very opposite of the parsimony principle. In fact, to use deterministic grammars for equations discovery we must employ external regularization methods to express the parsimony principle [5]. We visualize this comparison in the right-hand side of Figure 2.2, which depicts the probability of generating a tree with height h for different values of p (the probability of the recursive production in G_L). Although equation discovery algorithms that employ deterministic grammars (i.e., LAGRAMGE) do not use probabilistic sampling, we assume uniform sampling of parse trees up to a given height, for illustrative purposes. We can see that the probability of sampling a tree with height h falls exponentially for PCFGs, while rising exponentially for CFGs.

2.2 PCFGs for Mathematical Expressions

We use grammars to constrain the search space of equations in equation discovery. To that end, we design specialized grammars for mathematical expressions that ensure correct mathematical syntax and express the desired types of mathematical expressions through the probability distributions of different variables, operators and functions in the production rules of a PCFG. We discuss several concepts important to designing such grammars before considering examples of PCFGs for mathematical expressions.

2.2.1 Ambiguity

An important concept to consider when working with grammars is ambiguity. A grammar is formally ambiguous if sentences exist that can be described by more than one parse tree, generated by the grammar. Grammars for mathematical expressions can express another type of ambiguity, called semantic ambiguity. All but the simplest mathematical expressions can be written in many mathematically equivalent, but grammatically distinct ways. This is mainly due to the distributivity, associativity and commutativity properties of the basic operations. For example, consider the many ways the expression x^2+xy can be written: $x\times x+x\times y = x\times y+x\times x = x\times(x+y) = x\times(y+x) = (x+y)\times x = (y+x)\times x$. Each results in a distinct parse tree. It is generally useful to adopt a canonical representation that each generated equation is converted into. This allows us to compare expressions to each other and check whether they are mathematically equivalent in addition to comparing their parse trees. In our work, we use the Python symbolic mathematics library SymPy [52] to simplify expressions and convert them into a canonical form, as well as to compare expressions symbolically.

2.2.2 Variables in PCFGs for mathematical expressions

When addressing an equation discovery task, we compose candidate expressions using a set of variables that is part of the definition of the problem. In a grammar, variables are part of the set of terminal symbols. The simplest way to define them is through a dedicated production rule, such as V in grammar G_L in Equation (2.4), which imposes a uniform

distribution over the variables. However, when we have significant background knowledge to leverage, we can use arbitrary probability distributions over variables, or even place variable-generating productions in different places in the structure of the grammar. In this work, we use the simpler option of a single production with a uniform distribution over variables.

2.2.3 Numerical constants in PCFGs for mathematical expressions

We represent numerical constants in a PCFG with a single terminal symbol, typically c. After generating an expression string, we simplify and enumerate all numerical constants that appear in it. For instance, if the grammar generates the string $c * c * x + c$, we transform it into $c_1 * x + c_2$. We estimate the optimal values of numerical constants during the parameter estimation step of equation discovery. Since too many constants slows down computation and can lead to overfitting, and too few constants can preclude the success of equation discovery, it is important to pay attention to the probability of generating constants when designing grammars for mathematical expressions.

2.2.4 Examples of general-purpose grammars

So far we considered one of the simplest examples of grammars for mathematical expressions, the linear grammar G_L, which generates sums of two variables. Due to its simplicity, the linear grammar is convenient to demonstrate certain mathematical properties of PCFGs, however, it is not a useful grammar in practice. In this section, we introduce three grammars for mathematical expressions with increasing degrees of complexity that can be used in practice for equation discovery. The three grammars are general-purpose grammars, since the only background knowledge they encode lies in the type of expression they can generate. For brevity, we present the grammars as CFGs, with the understanding that we can easily turn them into PCFGs by equipping their production rules with probabilities. The first of these is the polynomial grammar G_P, which generates polynomials:

Polynomial grammar G_P:		**Examples of generated expressions**
$P \to P + c * M \mid c * M \mid c$		$c_1 x^2$
$M \to M * V \mid V$	(2.24)	$c_1 x^3 + c_2$
$V \to x_1 \mid x_2 \mid \ldots \mid x_m.$		$c_1 x^5 + c_2 x^4 + c_3 x^2 + c_4 x$

In this grammar, the nonterminal P represents the concept of a polynomial (e.g., $x^3 - x^2 + 3$) and the nonterminal M represents monomials (e.g., $2x^3$). It is easy to extend the polynomial grammar into the rational grammar G_R, which can generate any rational function:

Rational grammar G_R:		**Examples of generated expressions**
$R \to (P) \ / \ (P)$		$\dfrac{c_1 x_2}{x_1^2}$
$P \to P + c * M \mid c * M \mid c$		
$M \to M * V \mid V$	(2.25)	$\dfrac{x_3}{c_1 x_1 + c_2 x_2^2 + c_3 x_1 x_3}$
$V \to x_1 \mid x_2 \mid \ldots \mid x_m.$		$\dfrac{x_2 + c_1}{c_2 x_1^2 + c_3 x_1}$

This grammar constructs a rational function by simply dividing two polynomials. Finally, we take a look at the universal mathematical grammar G_U, which can generate any mathematical expression, composed of the four basic operations:

Universal mathematical grammar G_u: **Examples of generated expressions**

$$\begin{aligned} &E \to E + F \mid E - F \mid F \\ &F \to F * T \mid F \,/\, T \mid T \\ &T \to (E) \mid V \mid c \\ &V \to x_1 \mid x_2 \mid \ldots \mid x_m. \end{aligned} \qquad (2.26) \qquad \begin{aligned} &x_3 - x_1 \\ &x_1 - \frac{c_1 x_2}{x_1} \\ &c_1 x_1 \left(-\frac{x_1 x_2}{x_3} + x_3 \right) \end{aligned}$$

In the universal grammar G_U, the nonterminal E represents an expression in the form of a sum or difference of two factors, F is a factor obtained by multiplying or dividing two terms and T is a term, which can be either a variable, a numerical constant or another expression.

2.2.5 Special functions in grammars for mathematical expressions

The three grammars above are limited to expressions, composed using the operations $+, -, *, /$. We can include other mathematical functions by explicitly including them in the grammar. Such functions can be:

- **power** $x^2, x^3, \ldots,$
- **roots** $\sqrt{}, \sqrt[3]{}, \ldots,$
- **the exponential function and the logarithm** $\exp, \log,$
- **trigonometric functions** $\sin, \cos, \tan,$
- **inverse trigonometric functions** $\arcsin, \arccos, \arctan,$
- **hyperbolic functions** $\sinh, \cosh, \tanh,$
- etc.

Note that some of these functions can be generated by the grammar using other operations, i.e., $x^3 = x * x * x$ and $\tan x = \frac{\sin x}{\cos x}$. The selection of special functions to include in a grammar depends on the equation discovery task and is another way to express background knowledge. Since every additional function increases the search space, it is important to include only the functions we really deem necessary. Equation (2.27) below presents the universal mathematical grammar, extended with a set of special functions and equipped with probabilities. In the remainder of this work, this grammar will be our go-to general-purpose grammar for equation discovery, when the background knowledge is insufficient to construct a more specialized grammar.

Universal grammar G_U:

$$
\begin{aligned}
&E \to E + F\ [p_{\text{sum}}] \mid E - F\ [p_{\text{sub}}] \mid F\ [1 - p_{\text{sum}} - p_{\text{sub}}] \\
&F \to F * T\ [p_{\text{mul}}] \mid F\ /\ T\ [p_{\text{div}}] \mid T\ [1 - p_{\text{mul}} - p_{\text{div}}] \\
&T \to V\ [p_{\text{var}}] \mid c\ [p_{\text{con}}] \mid R\ [1 - p_{\text{var}} - p_{\text{con}}] \\
&R \to (E)\ [p_{\text{rec}}] \mid f_1(E)\ [p_{f_1}] \mid \ \ldots\ \mid f_k[p_{f_1}] \\
&V \to x_1\ [p_{x_1}] \mid x_2\ [p_{x_2}] \mid \ \ldots\ \mid x_m\ [p_{x_m}],
\end{aligned}
\tag{2.27}
$$

where $f_i \in \mathcal{F} = \{\sqrt{}, \exp, \log, \sin, \cos, \tan, \sinh, \cos, \tanh, \arcsin, \arccos, \arctan\}$,

$$p_{f_i} = \frac{1 - p_{\text{rec}}}{|\mathcal{F}|},$$

$$x_i \in \mathcal{V}; \quad p_{x_i} = \frac{1}{|\mathcal{V}|}$$

2.3 Search Space Visualization

In the context of equation discovery, we use grammars to encode background knowledge and to define the search space of mathematical expressions. In this chapter, we introduce several ways of studying and analyzing PCFGs, which can aid in grammar design. Here, we introduce a novel way of visualizing the space mathematical expressions, defined by a grammar or any other generator of mathematical expressions. Visualizing, analyzing and comparing the search spaces of different grammars can allow us to better understand their properties and help us design the best expression generator for our task.

2.3.1 Aggregated expression trees

Any mathematical expression can be represented as an expression tree, with mathematical operations and symbols as nodes. For instance, the expression $x + y$ has a node "Add", representing addition, with two children: x and y. We make use of this representation by introducing a novel concept, aggregated expression trees (AETs). AETs are trees, constructed by aggregating a large, hopefully representative, sample of expression trees generated by a grammar or another type of expression generator.

An AET is an expression tree with a special label in each node and edge which counts the number of occurrences of that node or edge among the expressions, aggregated into the AET. We initialize an AET as an empty expression tree, then add the individual expression trees to it one by one. Before adding an expression tree to the AET, we modify the labels of its nodes recursively, starting at the root node and proceeding towards the leaves of the expression tree. We relabel each node v, so that it contains the information on every node along the path from the root node to node v. For example, the node x in the expression tree of $x + y$ is labeled *Add_ x*, whereas the z in the expression $x + y * z$ is labeled *Add_ Mul_ z*. Finally, we add the renamed expression tree to the AET by modifying the counters in the nodes and edges of the AET accordingly: when a node label is not yet included in the AET, we add it to the AET, along with the edge to its parent, and initialize their counters to one. When a node identifier is already present in the AET, we increase its counter by one.

In this way, an aggregated expression tree encompasses all subtrees present in the collection of expression trees, while keeping track how many times each node and edge appears in the collection. Consider the example in Figure 2.3. The first image depicts the expression tree for $x + y$. For the purpose of generality, every expression tree has the root node *sys*, corresponding to the most general class of models we want to handle: systems of equations. Accordingly, the next node in the tree is *eq0*, corresponding to the first (and

only) equation in the system, followed by the familiar *Add*, x and y. The expression tree for $x + y + y$ is similar, featuring an additional branch. The AET, composed of the two expression trees, features four nodes in full opacity, which are shared by the two expression trees: *sys*, *eq0*, *Add* and x. The rest of the AET is semi-transparent, as the nodes and branches appear in only one out of the two expression trees in the collection.

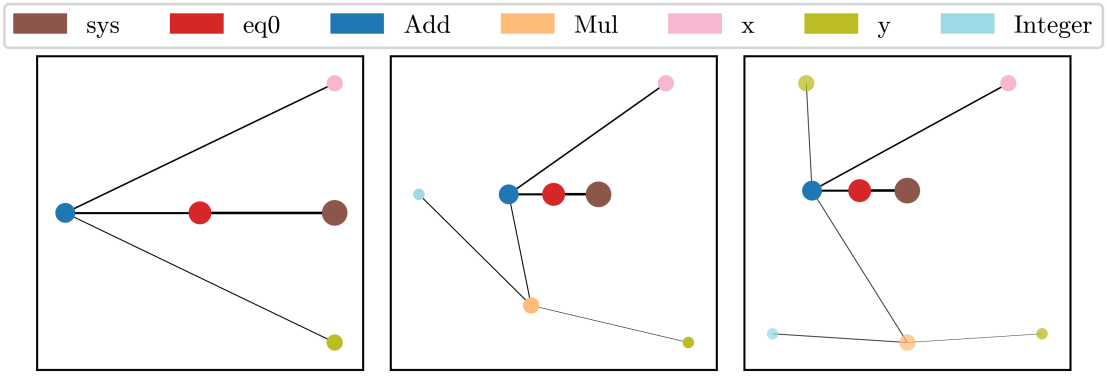


Figure 2.3: Example of building an aggregated expression tree: a) the expression tree of $x + y$, b) the expression tree of $x + y + y$, c) the aggregated expression tree. The size of nodes is inversely proportional to the height of the node in the tree, while the transparency of nodes and edges corresponds to their relative frequency in the collection of expression trees the AET was built from. Two special nodes are included in all three trees: *sys*, which is the root node of any expression tree and corresponds to a system of equations, and *eq0*, indicating the first equation from a system of equations.

As the next example, we can once again consider the linear grammar:

$$\begin{aligned} E &\to E + V \ [p_{\text{rec}}] \mid V \ [1 - p_{\text{rec}}] \\ V &\to x \ [p_{\text{var}}] \mid y \ [1 - p_{\text{var}}]. \end{aligned} \tag{2.28}$$

The grammar is parametrized by two probabilities: p_{rec}, the probability of recursion, and p_{var}, the probability of the variable x, with the default values $p_{\text{rec}} = 0.5$ and $p_{\text{var}} = 0.5$. By varying the values of these probabilities, we obtain versions of the linear grammar that generally define the same space of all mathematical expressions, but impose different probabilities, i.e., soft constraints, on the space. We can gain insight into the properties of these spaces through their aggregated expression trees, constructed by generating many random expressions using each version of the grammar and aggregating them.

In Figure 2.4, we present six AETs, obtained by randomly sampling 100 expressions using each version of the linear PCFG. The first row shows how the space of expressions changes as we vary the probability of recursion from 0 to 0.9, while keeping p_{var} at 0.5. When p_{rec} is zero, the grammar can generate only the expressions x and y and the corresponding AET is simple. For $p_{\text{rec}} = 0.9$, the grammar generates expressions of the form $nx + my$ for many different values of the integers n and m. For $p_{\text{rec}} = 0.5$, the number of unique integer values is lower, as choosing the recursive production is less likely. These observations are in close agreement with the effects of varying the probability of the recursive production we observed in Figure 2.2. This set of three AETs is another way of demonstrating and visualizing how the parsimony principle is parametrized by the probability of recursion.

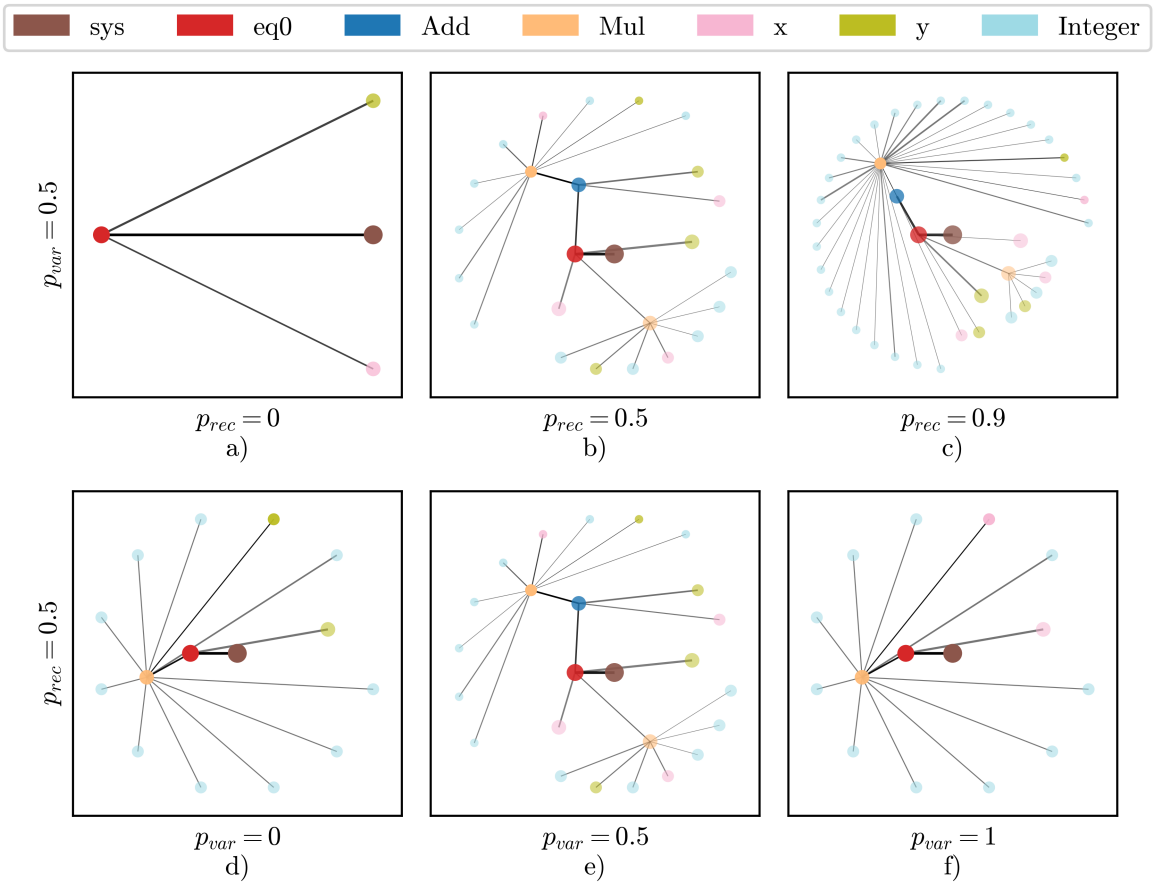


Figure 2.4: The aggregated expression trees for the linear grammar from Equation (2.28) with different values of production rule probabilities, obtained by aggregating 100 randomly sampled expressions using each set of probabilities. AETs a-c were generated by setting $p_{\text{var}} = 0.5$ and p_{rec}: a) 0, b) 0.5, c) 0.9. Meanwhile, for AETs d-f, p_{rec} was set to 0.5 and p_{var} was: d) 0, e) 0.5, f) 1. The size of nodes is inversely proportional to the height of the node in the tree and the transparency of nodes and edges corresponds to their frequency in the generated sample of expression trees.

To obtain the second row of AETs in Figure 2.4 we vary the distribution of variables, while keeping the probability of recursion at 0.5. Here, the visual differences between AETs are not as obvious. For $p_{\text{var}} = 0$ and $p_{\text{var}} = 1$, the grammar generates expressions of the form nx or ny, with the only two nodes in each AET that represent variables being exclusively x or y, respectively. In the AET of a grammar with a uniform variable distribution ($p_{\text{var}} = 0.5$), nodes of both variables appear in a tree, and the AET is more complex. This set of three AETs demonstrates how more other production rule probabilities affect parsimony and the shape of the space of expressions. More balanced distributions of production rules tend to generate a larger space of expressions than extremely biased distributions.

We will use the methodology of aggregated expression trees, introduced in this subsection, throughout the rest of this thesis to help us visualize spaces of mathematical expressions and understand the properties of different grammars.

2.4 Theoretical Analysis

So far, we introduced probabilistic context-free grammars for mathematical expressions and demonstrated how they inherently and intuitively parametrize the parsimony principle on the simple example of a linear grammar. In order to analyze the properties of practical grammars for mathematical expressions in the context of equation discovery, we will study the number of parse trees we have to sample from a given grammar in order to reconstruct a specific equation. We first investigate the differences between deterministic grammars and their probabilistic counterparts and then look into the impact that varying the production rule probabilities of a grammar can have on the performance of the equation discovery algorithm.

2.4.1 The Feynman symbolic regression database

We analyze and evaluate probabilistic grammar-based equation discovery using a publicly available benchmark – the Feynman database for symbolic regression [13]. The database consists of 100 equations from the three-volume course Feynman's Lectures on Physics, which cover classical mechanics, electromagnetism, quantum mechanics and other core topics from physics. The collection of equations prioritizes the most complex algebraic equations, i.e., those that do not involve derivatives or integrals. Each problem has between 1 and 9 variables and can contain the elementary operations $+, -, *, /$, as well as any of the special functions sqrt, exp, log, sin, cos, arcsin and tanh. The numerical constants that appear in the equations belong to the set of rational numbers, with the exception of e and π. The data was generated by numerically simulating the equations. The metadata available for each problem consists of the variable symbols and their corresponding physical units, which can be composed with 5 basic units: meter m, second s, kilogram kg, Kelvin K, and Volt V. To illustrate the types of problems, represented in the database, we present a few included equations. One of the simplest examples is the equation for the force of friction F:

$$F = \mu N,$$

where μ is the friction coefficient and N the normal force. As an example of a more complex equation, consider the relativistic momentum p of a particle:

$$p = \frac{mv}{\sqrt{1 - \frac{v^2}{c^2}}},$$

where m is the particles mass, v its velocity and c the speed of light in a vacuum. Finally one of the most complex equations in the database:

$$P = \frac{2\pi E_f p_d t}{h} \frac{\sin^2\left(\frac{t(\omega-\omega_0)}{2}\right)}{\left(\frac{t(\omega-\omega_0)}{2}\right)^2},$$

where P represents the probability of a state transition in an ammonium molecule, E_f is the electric field, p_d is the electric dipole, t is the time interval, h is the Planck constant, ω is the frequency and ω_0 the resonant frequency. The full collection of equations is provided in Appendix A.

2.4.2 Expected number of parse trees

The metric we use to analyze grammars in this section is the expected number of parse trees that we need to sample from a given grammar, so that the sample includes a parse tree,

corresponding to the mathematical expression we seek. The expected number depends on both the target expression and the grammar. We model the number of sampled parse trees as a random variable N and compute its expected value $E[N]$. The computation differs for probabilistic and deterministic grammars.

We can model the sampling of parse trees from a PCFG as consecutive trials in a Bernoulli process. In each trial, we sample a parse tree from the PCFG and check whether it matches the target expression (or by optimizing its parameter values and computing the error-of-fit on the data in a realistic scenario). We denote the probability of the parse tree corresponding to the target equation with p. N, the number of parse trees sampled before finding the desired one, follows the geometric distribution:

$$P(N = n) = (1 - p)^{n-1}p. \tag{2.29}$$

In other words, the Bernoulli process consists of a sequence of $n - 1$ unsuccessful trials, where the sampled parse tree does not correspond to the target (hence, the probability of each trial outcome is $1 - p$), followed by a single successful trial with the probability of p. The expected number of trials until (and including) the first success is

$$E_{PCFG}[N] = \frac{1}{p}, \tag{2.30}$$

which corresponds to the intuition that the more probable the target equation, the fewer samples are needed to reconstruct it.

The simplest deterministic approach using context-free grammars for the task of equation discovery tests expressions defined by the grammar systematically, until it finds the correct one. To ensure parsimony, the algorithm starts with the parse trees with the lowest height and progressively moves towards higher trees only once it has tried all the expressions of a given height. In other words, if we denote the height of the correct parse tree with h, the algorithm generates all parse trees with heights at most $h-1$. Since the distribution of parse trees with a given height in a CFG is uniform, the algorithm will on average also consider half of the parse trees with height exactly h. The expected number of parse trees considered by a deterministic grammar G is therefore

$$E_{CFG}[N] = N_G(h - 1) + \frac{1}{2}n_G(h), \tag{2.31}$$

where $n_G(h)$ is the number of parse trees with height exactly h (Equation (2.6) with $A = S$) and $N_G(h - 1)$ is the number of parse trees with height up to and including $h - 1$ (Equation (2.7) with $A = S$).

We find the probability p and the height h for a given target expression by parsing with the grammar of interest. We make use of the inside chart parser, a type of bottom-up algorithm, which recognizes lower-level elements before higher-level structures, as implemented in the Python package NLTK. If the given expression is derived by a given CFG, the parser finds the parse tree(s). In the case of PCFGs, the parser also computes the probability of the parse tree.

2.4.3 Probabilistic vs. deterministic grammar

Using the formulas in Equations (2.30) and (2.31), we calculate the expected number of sampled parse trees necessary to reconstruct each of the one hundred equations from the Feynman database. For different values of N, we are interested in how many of the target equations from the Feynman database we can expect to reconstruct by generating at most N sample parse trees from a given grammar. We can write this as $\frac{1}{100}\sum_{i}^{100} I(n \geq E[N_i])$,

where N denotes the number of sampled parse trees, $E[N_i]$ is the expected number of sampled trees necessary to reconstruct the i-th equation from the Feynman database, and $I(b)$ is a function indicating the truthfulness of the Boolean expression b, having the value of 1, if b is true, and 0 otherwise.

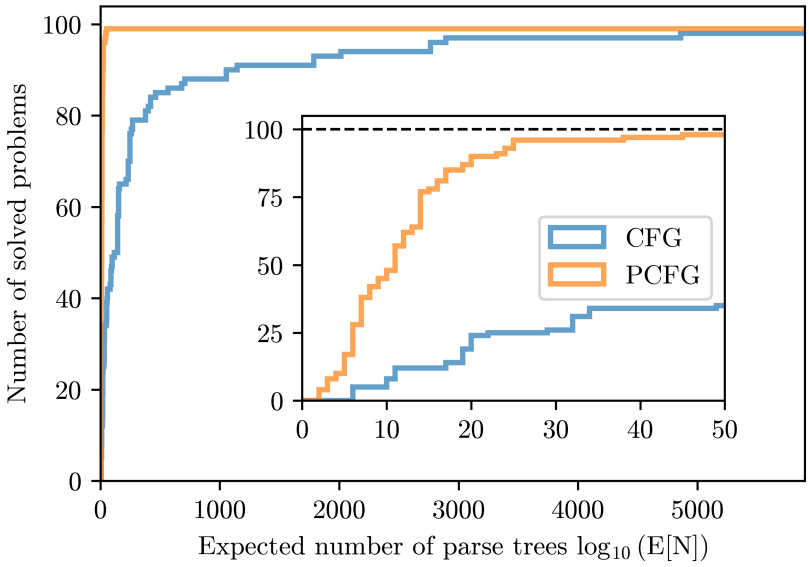


Figure 2.5: The number of problems from the Feynman symbolic regression database that we can expect to reconstruct by sampling a given number of parse trees from the universal mathematical PCFG (red line) in Equation (2.27) and its CFG counterpart (blue line). The inset provides a zoom-in on the range of the expected number of sampled parse trees below 10^{50}.

We compare the expected number of expressions to sample for the universal mathematical PCFG in Equation (2.27) and its CFG counterpart. The comparison, depicted in Figure 2.5, demonstrates that the expected number of samples for the deterministic grammar is many orders of magnitude higher than the expected number of samples for the probabilistic grammar. More specifically, we need to sample over 10^{4000} parse trees from the deterministic grammar in order to reconstruct all the equations from the Feynman database. In contrast, the probabilistic grammar requires, on average, less than 10^{50} samples. Half of the problems from the Feynman database can be successfully reconstructed when sampling around 10^{10} parse trees from the probabilistic grammar, while roughly 10^{60} samples are required for the same result using the deterministic version of the grammar.

The number of unique parse trees for a context-free grammar undergoes a super-exponential increase as the height of the parse tree increases. This holds true for both probabilistic and deterministic grammars. Despite this, the expected number of sampled expressions can vary greatly, spanning tens or even hundreds of orders of magnitude. The reason for this is that the probability distribution over parse trees produced by a deterministic grammar is uniform, whereas probabilistic grammars introduce a preference for simpler parse trees, as detailed in Section 2. The findings depicted in Figure 2.5 reveal that the bias introduced by probabilistic grammars closely aligns with the equations featured in the Feynman database, which signifies a connection between the bias and equations commonly used in science. In the subsequent section, we will demonstrate how a precise adjustment of the probabilities assigned to individual production rules in the grammar can shift the bias towards equations in the Feynman database even further.

2.4.4 Biased vs. unbiased probabilistic grammar

In the remainder of this work, we focus our attention on probabilistic grammars. In this subsection, we investigate the effect that modifying the prior distribution over parse trees by adjusting the probabilities of production rules has on the expected number of sampled trees. As seen in Figure 2.2, reducing the probabilities of recursive production rules leads to a preference for simpler equations. To extend this analysis, we examine the impact of varying the probabilities of the other production rules on the expected number of sampled trees. In order to maintain a straightforward and clear comparative analysis, we vary the following four parameters:

1. the ratio between the probabilities of summation and subtraction:
   $r_{\text{sum}} = \frac{P(E \to E+F)}{P(E \to E-F)}$,
2. the ratio between the probabilities of multiplication and division:
   $r_{\text{mul}} = \frac{P(F \to F*T)}{P(F \to F/T)}$,
3. the ratio between the probabilities of a constant and a variable:
   $r_{\text{const}} = \frac{P(F \to V)}{P(F \to c)}$,
4. the ratio of the total probability of the set of special functions and no special function:
   $r_{\text{funct}} = \frac{\sum_k P(R \to f_k(E))}{P(R \to (E))}$.

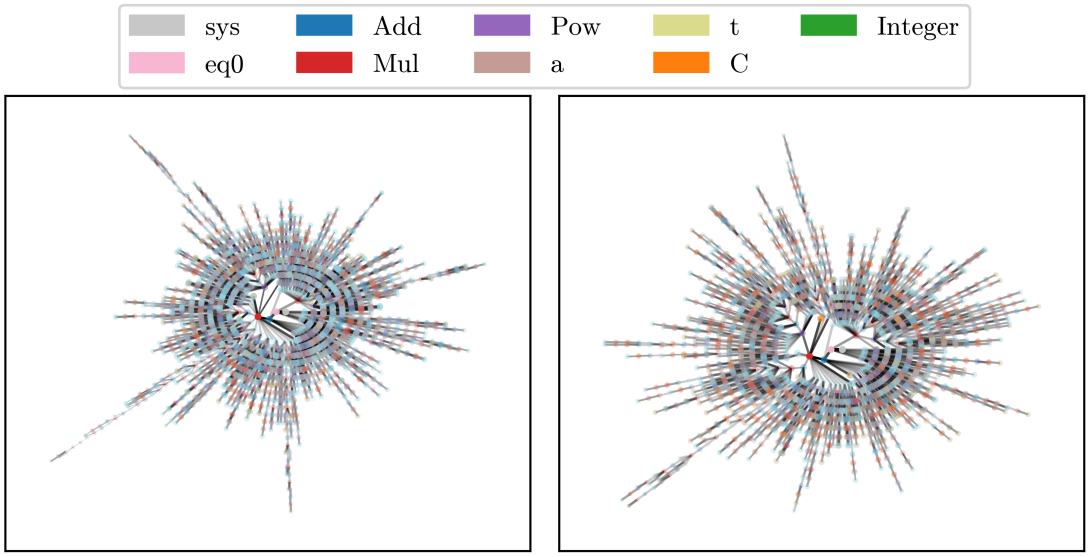
In our analysis, we begin with a grammar that sets all of the ratios to one, which we term the uniform grammar. This grammar does not favor any particular operator, function, or type of atomic term (variable or constant) in an equation, and therefore is devoid of any inherent biases. On the other hand, the biased grammar introduces intuitive preferences that are frequently used by scientists and engineers.

The precise parameter settings for both grammars are listed in Table 2.1.

Table 2.1: Parameter values for the uniform and the biased universal grammars.

	r_{sum}	r_{mul}	r_{const}	r_{funct}
Uniform	1	1	1	1
Biased	0.4	1.5	0.25	0.67

First, we can visualize the space of expressions, spanned by the uniform and biased universal grammars by sampling 1000 random expressions from each and constructing aggregated expression trees in Figure 2.6. Visually, we can see that the AET of the biased grammar is smaller, indicating a more constrained space of expressions. We can quantify the difference by considering the number of nodes in each AET: 5724 for the universal grammar and 5278 for the biased grammar. The difference is small, but not insignificant. Next, we focus on the expected number of parse trees needed for successful equation discovery, obtained by parsing the expressions from the Feynman database. The outcomes of the comparison are displayed in Figure 2.7. The left-hand histogram exhibits that incorporating the bias into the probabilistic grammar leads to a significant decline in the expected number of samples, although it is not as substantial as the one witnessed when comparing the probabilistic and deterministic grammars. In fact, the biased grammar lowers the anticipated number of samples for 86 out of the one hundred Feynman equations, with only ten equations showing an increase (while the expected number of samples for four



Uniform: 5724 nodes
a)

Biased: 5278 nodes
b)

Figure 2.6: Aggregated expression trees for: a) the uniform universal PCFG and b) the biased universal PCFG. The AETs were constructed by randomly generating 1000 expressions with each grammar. The size of nodes is inversely proportional to the height of the node in the tree, while the transparency of nodes and edges corresponds to the relative frequency of the nodes and edges in the collection of expression trees. Additionally, we provide the number of nodes in each AET in its label.

equations remains the same for both grammars). Among the 86 cases with a reduction in the expected number of samples, 58 fall within the range of a 25% decrease in the expected number of parse trees at most, and a 90% decrease at best. For 28 target equations, the anticipated number of models is lowered by more than one order of magnitude, with the largest decrease reaching seven orders of magnitude.

The biased grammar appears to utilize the fundamental principle of parsimony in a more sophisticated manner. One might assume that the biased grammar would lead to a more significant reduction in the expected number of samples for simpler target equations. However, the scatter plot on the right-hand side of Figure 2.7 indicates the opposite trend: the reduction generally increases as the equation complexity grows, with the Spearman correlation coefficient between the reduction rate and complexity being 0.5. In this plot, we measured the equation complexity as the length (i.e., the number of alphanumerical characters) of its textual representation.

To conclude, our comparison between a uniform and a biased universal grammar shows that manipulating the production probabilities of a probabilistic grammar can reduce the expected number of samples by more than one order of magnitude, particularly for complex equations. This highlights the potential of probabilistic context-free grammars as a flexible means of encoding prior knowledge and domain expertise in equation discovery. In the next section, we present empirical evidence to support this assertion.

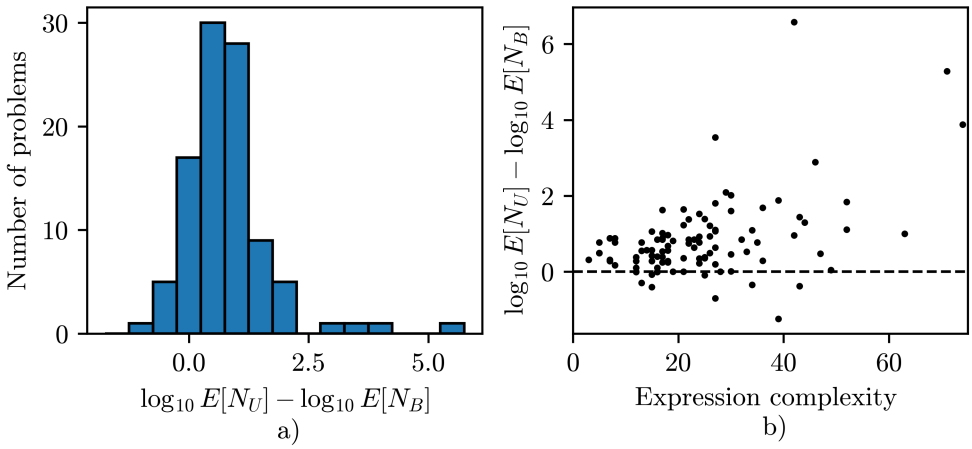


Figure 2.7: Reduction in the expected number of parse trees needed to reconstruct the one hundred equations from the Feynman database, induced by introducing bias into the probabilistic grammar for mathematical expressions. Depicted: a) histogram of the number of Feynman equations (y-axis) with a given reduction factor (x-axis), b) scatter plot of the reduction factor (y-axis) and equation complexity (x-axis) for each Feynman equation. $E[N_U]$ and $E[N_B]$ indicate the expected number of sampled parse trees for the uniform and the biased universal grammar, respectively.

2.5 Empirical Analysis

To complement our theoretical analysis, we have created a straightforward algorithm using probabilistic context-free grammars in Python for discovering equations. Our algorithm leverages a Monte-Carlo method to sample from the probability distribution of all possible equations. Each equation discovery task comprises variable names and simulated or measured data. For each task, we provide the correct equation to compare with the results of the algorithm, which include the discovered equations. In our investigation of the algorithm's performance, we evaluate its output against the correct equation for each task.

To tackle each equation discovery task, we generate a universal mathematical probabilistic context-free grammar (PCFG) that encompasses the measured variables and a constant symbol as terminal symbols. We then randomly sample a large pool of candidate equation structures from the PCFG and assess their validity against the data in the Feynman database. This process requires choosing suitable values for the constants in the equation to optimize the fit to the data. The most appropriate equation that matches the data is identified and returned as the output. Alternatively, one could continue sampling candidate equation structures until finding a suitable equation that fits the data well, with the degree of satisfaction specified beforehand.

2.5.1 Monte-Carlo sampling algorithm

The Monte-Carlo approach to grammar-guided equation discovery is outlined in Algorithm 2.2. To generate candidate expressions for the equations' right-hand side, we use the procedure GENERATE_SAMPLE from Algorithm 1, which samples them independently. Our implementation of the algorithm in Python leverages the Natural Language Toolkit (NLTK) [53] to work with grammars. We then process each sampled expression e using SymPy [52], a Python library for symbolic mathematics, to obtain its canonical form e_c and determine the list of constant parameters it contains. To estimate the constant pa-

Algorithm 2.2: DISCOVER_EQUATIONS(G, A)
Monte-Carlo algorithm for grammar-based equation discovery.

Data: Probabilistic grammar $G = (\mathcal{N}, \mathcal{T}, \mathcal{R}, S)$ generating mathematical expressions, number of samples N, data set D, target variable v
Result: List of equations *eqns*, sorted according to increasing error on D

```
initialize eqns = [ ];
for i = 1, i ≤ N do
    (e, p) = GENERATE_SAMPLE(G, S);
    e_c = CANONICAL_FORM(e);
    eqn = FIT_PARAMETERS(e_c, v, D);
    error = ReRMSE(eqn, D);
    eqns.append(eqn, p, error);
end
return eqns.sort(key=error, order=increasing);
```

rameter values, we minimize the root mean squared error (RMSE) of the equation $v = e_c$ on the dataset D:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{2.32}$$

where n is the number of samples in the data, y_i is the i-th value of the independent variable in the data, and $\hat{y}_i$ is the value, predicted by our equation. As the minimization algorithm we use differential evolution (DE) [54], an efficient and widely-used method for global optimization, with DE parameters set similar to those reported by Lukšič [55]. To facilitate comparisons between different equation discovery problems, we use as the final score the relative root mean squared error ReRMSE, which is RMSE, normalized by the standard devation of the data:

$$\text{ReRMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{2.33}$$

where $\bar{y}$ is the mean value of y in the data. The algorithm reports the sampled equations sorted by increasing ReRMSE.

2.5.2 Empirical setup

To verify our theoretical analysis experimentally, we utilized the Monte-Carlo algorithm for equation discovery on the Feynman database, which contains one hundred equations from physics. We evaluated the performance of the uniform and biased versions of the universal mathematical grammar described in Equation (2.27) with production probability distributions that are parametrized as shown in Table 2.1. For each Feynman equation, we generated 10^5 candidate expressions using Algorithm 1 with each of the studied grammars. Since we had practical constraints and time limitations, we only performed parameter estimation for equations that had at most five constant parameters. Equations with more than five constant parameters were deemed inadmissible.

We improved the computational efficiency of the equation discovery process by identifying and removing duplicate expressions. Specifically, we checked the generated expressions

Table 2.2: Summary of experimental results on reconstructing the hundred target equations from the Feynman database using the Monte-Carlo algorithm for grammar-based equation discovery with the uniform and biased versions of the universal grammar for mathematical expressions.

Experiment	N unique [$\cdot 10^3$]	Coverage	Reconstructed equations
uniform 1	31 ± 4	0.36 ± 0.04	36
uniform 2	31 ± 4	0.36 ± 0.04	38
uniform 3	28 ± 7	0.32 ± 0.08	35
biased 1	31 ± 4	0.51 ± 0.04	37
biased 2	30 ± 5	0.49 ± 0.07	37
biased 3	28 ± 7	0.46 ± 0.10	36

for duplicates, i.e., expressions with identical canonical forms, and estimated the parameters for all expressions sharing the same canonical form only once. Additionally, since the Feynman database contains noise-free data, correct equations tend to have low error. In our experiments, we used a threshold value of 10^{-9} for the relative root mean squared error (ReRMSE) to determine whether a candidate equation is a correct reconstruction of the target equation in the Feynman database. A target equation from the Feynman database is considered successfully reconstructed with a given grammar if the corresponding sample contains at least one matching candidate equation. To account for the stochastic nature of the algorithm, we performed three independent runs of the Monte-Carlo sampling for each grammar.

2.5.3 Results

The empirical results are summarized in Table 2.2, with detailed results for each Feynman equation reported in Appendix B. The first column of the table shows the average number of unique expressions sampled per target equation across all the problems in the Feynman database. Due to the semantic ambiguity of the grammars for mathematical expressions, the reported numbers are much lower than the total number of sampled expressions. Specifically, only around 30% of the samples correspond to unique canonical expressions for both the uniform and biased grammar.

The second column of Table 2.2 reports the coverage achieved by the Monte-Carlo algorithm in terms of the total probability of the sampled expressions, which is the sum of the probabilities of their corresponding parse trees. We observe that sampling with the biased grammar covers about half of the space of candidate equations in terms of total probability, while the uniform universal grammar achieves a coverage of only about 0.35. This difference can be attributed to changes in the structure of the search space brought about by the different probability distributions of production rules. The probability distributions for the biased grammar are generally more varied, with some rules having higher probabilities than others, in contrast to the uniform grammar that assigns equal probabilities to all rules. This effect is also reflected in the probability distributions over parse trees, with a few parse trees contributing most of the total covered probability, while the majority of parse trees making only a minuscule contribution. Additionally, parse trees with a higher contribution to coverage are more likely to be sampled, suggesting that coverage can be interpreted as a measure of inequality among parse trees, which is related to the amount of information in the prior distribution. These observations align with the concept of uniform priors being the least informative ones, a concept widely used in Bayesian statistics [56].

The third column in our results table indicates the number of successfully reconstructed target equations from the Feynman database. Both grammars were able to reconstruct roughly 36 of the 100 equations after generating 10^5 samples. While both grammars demonstrated comparable levels of performance in terms of successful reconstructions, our theoretical analysis suggested that the biased grammar would perform better.

To contextualize our results, we can compare them to those reported by Udrescu [48]. Specifically, the AI Feynman approach was able to discover all 100 equations from the Feynman database, while the symbolic regression method Eureqa [6] was able to reconstruct 71% of the equations. In contrast, our probabilistic grammar-based approach reconstructed 37% of the equations in our experiments. It should be noted, however, that this lower performance was expected, given that the approach we presented was mainly intended for illustrative purposes and relied on a very simple algorithm. We will revisit this comparison in our analysis of the results.

2.5.4 Resampling

The results presented in Table 2.2 cannot be directly compared with the theoretical results because they only summarize performance at a fixed number of sampled expressions. A more general presentation of the results would show a performance curve, in other words, performance vs. sample size. The easiest way would be to simply take a sliding minimum across the error of the sampled expressions in the order they were sampled. However, the order is arbitrary - if we repeat the experiment, we would get the sampled expressions in an entirely different order (based on their probabilities) and a different performance curve. To alleviate the stochasticity of the algorithm, we employ resampling on the sample of expressions. The measure of success we are working with is the success ratio, which we define as the portion of successfully reconstructed equations from the Feynman dataset. We treat the sample of approx. 30000 unique canonical expressions with their corresponding probabilities as a probability distribution. From this distribution we then sample 30000 expressions, without replacement. In other words, we randomly reorder the set of sampled expressions, while taking into account the probability of each expressions. By repeating this many times (100 in our case), we can average the success ratio at each sample size. This value is the average success rate, depicted in Figure 2.8. It can be interpreted as the expected success rate at a given sample size. In other words, if the Monte-Carlo algorithm were repeatedly run many times, each time sampling N canonical expressions for each problem from the Feynman database, computing the average of the portion of solved problems would give a value close to the average success rate we report.

In our experiment, we perform three independent Monte-Carlo samplings for each of the two grammars, resulting in six sets of sampled expressions. The resampling procedure is performed separately for each of the six sample sets. Figure 2.8 depicts the minimum and maximum of the average success rate across the three samplings at each sample size for each of the two grammars.

2.5.5 Theoretical expectation of success rate

In the theoretical analysis in Section 3, we use an inside chart parser algorithm [57] to parse the target expression of each problem from the Feynman database, using either the universal or the biased universal grammar. We take the parsed tree probabilities as an approximation for the probability that a randomly sampled expression corresponds to the correct solution for a given problem. We can use the probabilities to calculate a theoretical expectation of the success rate (dependent on the sample size) for each grammar. The probability of finding the correct solution of a problem with index i in a sample of N

expression, generated with grammar G, is

$$P_{G,i}(N) = 1 - (1 - p_{G,i})^N \approx 1 - (1 - \widetilde{p_{G,i}})^N,$$

where $p_{G,i}$ is the probability of randomly generating the correct solution to problem i using grammar G. We approximate this probability with the parsed probability of a single parse tree $p_{G,i} \approx \widetilde{p_{G,i}}$. Finally we consider the complete Feynman dataset of one hundred equations to arrive at the expected success rate

$$E[\text{success rate}](N) = \frac{1}{100}\sum_{i=1}^{100} P_{G,i}(N) \approx 1 - \frac{1}{100}\sum_{i=1}^{100}(1 - \widetilde{p_{G,i}})^N.$$

Once we perform our sampling experiment, we can directly compare the empirical success rate with its theoretical expectation, which is depicted as the pair of full lines in Figure 2.8.

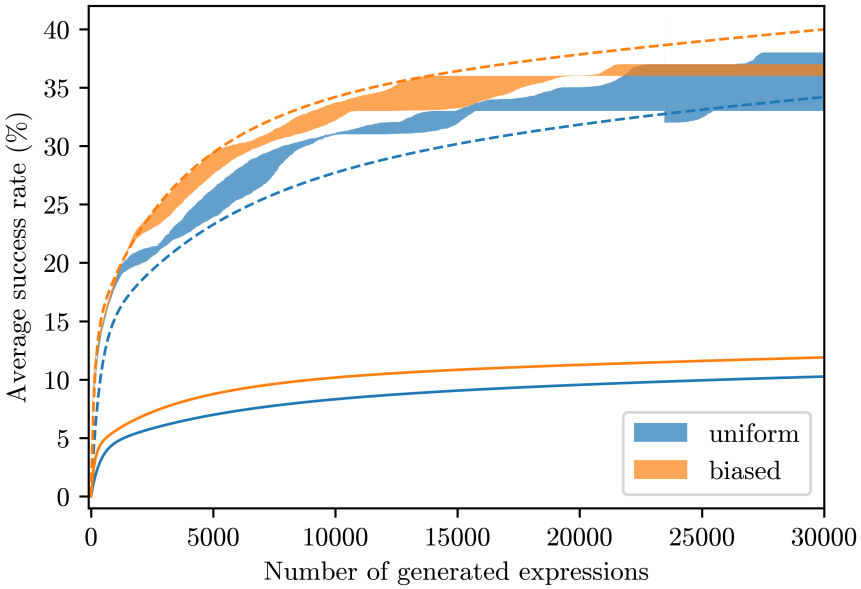


Figure 2.8: Average rate of successful reconstruction achieved with the uniform and biased grammar on the Feynman database. The filled regions represent empirical results across three independent runs of Algorithm 2. The solid lines correspond to the predicted success rates based on the analysis in Section 3.4. The dashed lines represent the predicted success rates, corrected by taking into account the empirically estimated level of semantic ambiguity for each grammar.

However, it should be noted that these theoretical success rates are much lower than the success rates obtained empirically. The reason for this discrepancy is the semantic ambiguity present in the universal grammar. While we know that the grammars generate many expressions that are mathematically identical, we approximate the sum of their probabilities by considering only a single parse tree. We estimate the ratio between these two values from our experiments as the ratio between the number of unique expressions in the sample and the full sample size (10^5), averaged over the three independent samplings.

- uniform: $\frac{N_{\text{unique}}}{N} = 0.299$,
- biased: $\frac{N_{\text{unique}}}{N} = 0.297$.

We apply these ratios to adjust the probability estimates for generating a parse tree that simplifies to a canonical expression representing the solution to the problem under study. This adjustment results in a decrease in the expected number of necessary samples for each grammar, as reflected by the dashed lines in Figure 2.8. The corrected theoretical predictions for the expected number of samples and success rates for each grammar closely match the empirical curves obtained by resampling the empirical results. The corrected predictions offer strong support for our theoretical analysis.

2.5.6 Analysis of the results

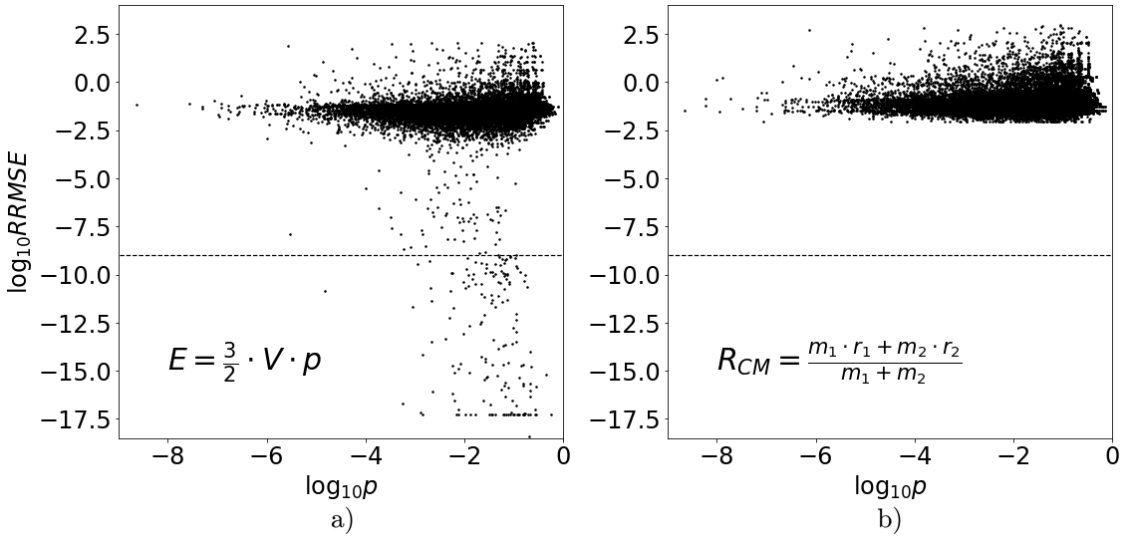


Figure 2.9: Scatter plots of the probability of a sampled expression against the error of the corresponding equation for two samples taken with the uniform universal grammar. The samples correspond to: a) a simple, successfully reconstructed target equation from the Feynman database, b) a more complex equation that was not successfully reconstructed. The dashed line represents our error threshold for considering a candidate expression to be correct. The best sampled expressions are found in the bottom right corner of each scatter plot – they have high probability and the corresponding equations have low error.

Overall, the performance of the Monte Carlo algorithm for grammar-based equation discovery is subpar. In our experiments, the method was able to solve only about 37% of the equation discovery tasks from the Feynman dataset. To better understand this limited performance, we need to examine the results for some specific tasks.

In Figure 2.9, we analyze one successfully solved equation discovery task and one unsuccessful one. We visualize the probability distribution of expressions and their errors as a scatter plot on a logarithmic scale. For both equation discovery tasks, the majority of sampled expressions are found in a cluster with moderate probability and high error. For the solved problem, several points are scattered across more than 15 orders of magnitude in error, with many of them falling below the error threshold for a correct expression. In the case of the unsolved problem, there are no points below the main cluster. The information in Appendix B (equations 20 and 39) indicates that the solved problem represents an easier task than the unsolved one. Equation #39 is less complex than #20 in all measures except the number of parameters. With an estimated generation probability of $3 \cdot 10^{-4}$, we can expect at least a few correct solutions for task #39 in a sample of 10^5 expressions, on

average. In contrast, the estimated probability of $9 \cdot 10^{-15}$ for equation #20 means that we would need to be extremely lucky to find the correct solution in our samplings.

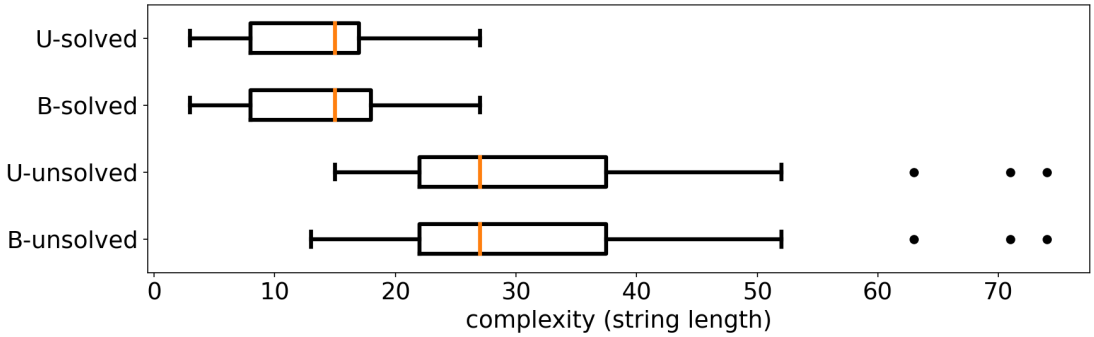


Figure 2.10: A box plot comparison of the complexity of equations from the Feynman database that were successfully reconstructed in the experiments (solved) with the complexity of equations that the algorithm was unable to reconstruct (unsolved). Depicted separately are experiments using the uniform universal grammar (labelled U) and the biased universal grammar (labelled B). The orange line indicates the median of the distribution, while dots indicate outliers.

Another high-level view of the results is presented in Figure 2.10, which compares the distributions of target expression complexity for the sets of solved and unsolved problems from the Feynman database. We observe that the majority of problems that the Monte-Carlo algorithm was unable to solve are more complex than the majority of solved problems; however, there is some overlap in the tails of the distributions.

We demonstrated how the nature of sampling a probabilistic grammar, along with our chosen prior distributions, biases the search towards simpler expressions. This bias allows the procedure to successfully discover solutions for the majority of less complex problems from the Feynman database. During the sampling procedure, however, we ignore all information on previously generated expressions' errors. To discover complex equations, intelligent and adaptive data-driven sampling algorithms must be developed for probabilistic grammar-based methods.

Finally, it should be noted that the complexity of a parse tree used to derive a particular mathematical expression is not directly related to the complexity of the expression, as demonstrated. The complexity of the parse tree is dependent on the grammar employed for equation discovery. For example, one grammar may be able to derive the expression using a parse tree with a height of only three, while another may require a parse tree with a height of ten. In this chapter, we exclusively use the universal grammar for mathematical expressions, given in Equation (2.27). However, utilizing alternative probabilistic grammars, such as those derived from deterministic grammars considered by Todorovski and Džeroski [5] that employ domain-specific or cross-domain knowledge about mathematical modeling to restrict the set of candidate equations, could greatly improve the efficiency of reconstructing the target equations from the Feynman database. Such alternative grammars may be able to derive the target equations with smaller parse trees, ultimately decreasing the number of samples required for successful reconstruction.

Chapter 3

Attribute Grammars for Dimensional Consistency

Equation discovery methods often face the challenge of working within an infinite space of possible equations. To address this, various methods have been employed to constrain the search space, such as limiting equations to linear expressions in their parameters [3], imposing complexity constraints, and constructing equations from limited sets of permissible terms [12]. Effective and meaningful constraints can be derived by leveraging domain knowledge, such as prior information regarding a dynamical system expressed through a process-entity formalism.

In many physical sciences, measurement units are a form of background knowledge that is readily available. Units impose strict constraints on the structure of equations, for example, variables can only be added or subtracted if their units are identical, and both sides of an equation must have the same units. Scientists have traditionally employed dimensional analysis as a tool for verifying the plausibility of equations and to help derive new equations.

The Buckingham Π theorem [45] offers a technique for transforming an equation into one that contains a reduced number of dimensionless combinations of variables. In the fields of system identification and signal processing, various methods rely on Buckingham's dimensionless Π groups to discover equations from data. However, methods based solely on dimensionless products are restricted in the types of expressions they can generate, and therefore need to be combined with other approaches to leverage additional forms of domain knowledge.

3.1 Existing Work on Dimensionally-Consistent Equation Discovery

The roots of dimensional analysis can be traced back to the concepts of similar systems developed by Newton, Galileo, and Fourier. In 1914, Edgar Buckingham synthesized these ideas into the concept of physically-similar systems [45], [58]. This concept formalized the notion that physical laws are independent of units of measurement and provided a means of reducing physical equations to their most general, dimensionless form. Since then, dimensional analysis (and the related scaling theory [59]) has played a crucial role in discovering physical laws across various fields, such as fluid dynamics [60], radiation [61], chemical reactions [62], biophysics [63], economics [64], astrophysics [65], among others.

Equation discovery and symbolic regression comprise a broad research area with various approaches. Some methods, such as genetic programming, do not utilize any background

knowledge but instead rely on powerful search algorithms to explore the vast space of all possible equations. Genetic programming represents mathematical expressions as expression trees and conducts an evolutionary search by applying mutations and recombinations to these trees [6]. Recently, deep learning-based approaches have shown potential in this area [41], [66]; however, these methods typically do not incorporate background knowledge.

In contrast to the previously discussed methods, some symbolic regression approaches focus on utilizing domain-specific knowledge to constrain the search space of equations [11]. SINDy [3] is a popular equation discovery tool that represents observed data as a linear combination of product terms and optimizes the associated numerical coefficients using sparse linear regression. The model is constructed using a pre-defined library of allowable data transformations, which is a limited way to incorporate background knowledge. ProBMoT [12] employs process-based modeling as a powerful framework for expressing domain knowledge and defining the search space.

Several methods leverage dimensional analysis to either solve the equation discovery problem or to reduce the search space and enhance their performance. COPER [67] uses dimensional analysis and the Buckingham Π theorem to examine a data set of observations, assess the variables and their measurement units for any extraneous or absent variables, and discover a restricted set of dimensionally-consistent equations by exploring the space of polynomial functions. ABACUS [23] employs the heuristic strategies of BACON [21], proportionality graphs, and suspension search to gradually build dimensionally-consistent equations. This method is useful when dimension-related information is not readily available, particularly in non-physical domains. SDS [68] overcomes this constraint by focusing on scale-types, whereby each variable is characterized as one of three types, imposing robust restrictions on the search space of equation discovery. Dimensional function synthesis is an approach that targets resource-efficient equation discovery for low-latency applications in embedded systems and data streams [47]. The method uses the Buckingham Π theorem to identify all dimensionless Π groups and merges them to form the final equation. One drawback of this method is that it can only generate a limited range of expression structures.

Dimensionally-aware genetic programming methods have been shown to improve the efficiency of genetic programming by introducing evolutionary pressure on dimensional incorrectness [46]. In this form of soft constraint in equation discovery, dimensional consistency is enforced during the search process. Alternatively, a different approach [43] employs context-free grammars to constrain the search space to dimensionally-consistent expressions, with hard constraints ensuring dimensional consistency of generated expressions through mutations and recombinations on derivation trees instead of expression trees. A recent study [69] analyzed the fitness landscape of dimensionally-aware genetic programming search spaces using a subset of equations from the Feynman symbolic regression database. The study showed that adding information about the variable dimensionality efficiently guides the search algorithm.

In the field of equation discovery research, it is widely recognized that utilizing domain-specific knowledge and dimensional analysis can greatly enhance the efficiency and elegance of approaches. However, effectively integrating dimensional consistency with other forms of background knowledge remains a challenging task for most methods. To address this issue, a recent work by Bakarji et al. [70] introduces a novel extension to the sparse regression package SINDy, which incorporates measurement units into the discovery process. By leveraging the Buckingham Π theorem, the approach identifies dimensionless groups, enabling sparse regression to discover dimensionless equations with fewer variables while preserving the expressivity of its library of data transformations.

In contrast to the aforementioned work, AI Feynman [13] takes a physics-inspired ap-

proach to equation discovery, which involves a series of modules that iteratively reduce the complexity of the problem until a polynomial fit or brute force search can be used to discover the equation. The first module focuses on dimensional analysis, which replaces the set of variables with a smaller set of dimensionless quantities. Other modules incorporate different types of cross-domain knowledge, such as symmetry and separability, but do not rely on domain-specific background knowledge.

A recent study by Crochepierre et al. [71] proposes an innovative approach to equation discovery using reinforcement learning. The method samples from a context-free grammar that encodes dimensionally-consistent mathematical expressions and other forms of domain knowledge. While the approach appears promising, it is currently unclear whether it provides any advantage over random sampling. Furthermore, the study's primary focus is on intelligent sampling, rather than on the representation of domain knowledge using grammars.

Deep Symbolic Optimization (DSO) is a state-of-the-art method for generating symbolic expressions using reinforcement learning, as proposed by [14]. DSO employs a recursive neural network trained with a novel risk-seeking policy gradient, which rewards best-case performance rather than expected performance. The method also utilizes genetic programming between RNN iterations to further improve performance. In addition, DSO can leverage limited background knowledge in the form of priors and constraints, including dimensional consistency, by directly adjusting the logits produced during sampling, as described in [36].

3.2 Dimensions and Measurement Units

Units and dimensions are important pieces of background knowledge that are commonly utilized by human scientists, but can be difficult to express using context-free grammars. In this context, we define $\mathcal{Q}$ as the set of symbols in an equation that correspond to quantities in the physical system being studied. Each quantity is associated with a specific dimension, such as length, time, or charge. Units of measurement, on the other hand, are realizations of dimensions and may not be uniquely defined, but rather chosen based on convention or utility.

To represent physical units, we can use vectors with integer elements, relative to a base that defines the fundamental units needed to describe the system of interest. For instance, consider the equation for accelerated motion, $x = \frac{1}{2}at^2$, where $\mathcal{Q} = x, t, a$ correspond to the dimensions of distance, time, and acceleration, respectively, with corresponding physical units $\mathcal{U} = m, s, ms^{-2}$. Using a unit base $\mathcal{U}_\prime = m, s$, we can represent the physical units as vectors in $\mathbb{Z}^2$: $u_x = (1, 0), u_t = (0, 1), u_a = (1, -2)$. A quantity with a unit of 1 in a given basis is called a dimensionless quantity, which we denote as $u_Q = \mathbf{0} = (0, 0, \ldots, 0)$. Quantities that are not dimensionless are referred to as dimensioned quantities.

When constructing mathematical expressions, the presence of dimensions and physical units can introduce constraints on their structure. Specifically, when adding or subtracting terms, the units of each term must be identical, denoted as $u_{Q_1 \pm Q_2} = u_{Q_1} = u_{Q_2}$. Conversely, when multiplying or dividing terms, the resulting units are given by the sum or difference of the units of the terms, respectively, denoted as $u_{Q_1 \cdot Q_2} = u_{Q_1} + u_{Q_2}$ and $u_{Q_1/Q_2} = u_{Q_1} - u_{Q_2}$.

We say that an algebraic expression is dimensionally-consistent if it adheres to these rules of unit arithmetic. For instance, the expression $vt + x_0$, where $u_v = ms^{-1}$, $u_t = s$, and $u_{x_0} = m$, is dimensionally-consistent, whereas the expression $vt + v_0$, where $u_v = u_{v_0} = ms^{-1}$ and $u_t = s$, is not, because the units of the terms in the sum are not equal. Similarly, we say that an equation is dimensionally-consistent if the expressions on either side of the

equality are dimensionally-consistent and have identical units. For example, the equation $x = vt$, where $u_x = m$, $u_v = ms^{-1}$, and $u_t = s$, is dimensionally-consistent, whereas the equation $a = vt$, where $u_a = ms^{-2}$, $u_v = ms^{-1}$, and $u_t = s$, is not dimensionally-consistent.

3.3 Probabilistic Attribute Grammars (PAGs)

We utilize probabilistic attribute grammars (PAGs), which are an extension of probabilistic context-free grammars that allow for the specification of attributes for each (nonterminal or terminal) symbol in the grammar. Attribute values can be defined through attribute rules associated with the production rules of the grammar, and can be used to express relationships between attributes of different symbols, as well as constraints on attribute values [72].

In our specific application, we assign a single attribute to each grammar symbol, which corresponds to the vector representation of the physical unit of the symbol. The attribute rules associated with each production rule express the arithmetic rules for dimensioned quantities, specifically the requirement that the units of terms in addition or subtraction must be identical. For example, by extending the polynomial probabilistic context-free grammar (PCFG) in Equation (2.24) with attributes and attribute rules, we arrive at the following dimensional attribute grammar:

$$\begin{array}{lcl} P & \rightarrow & P + c * M \ [p_P] \ \{P1.u = P2.u = M.u\} \quad | \quad c * M \ [1 - p_P] \ \{P.u = M.u\} \\ M & \rightarrow & M * V \ [p_M] \ \{M1.u = M2.u + V.u\} \quad | \quad V \ [1 - p_M] \ \{M.u = V.u\} \\ V & \rightarrow & a \ [p_a] \ \{V.u = a.u\} \quad | \quad t \ [1 - p_a] \ \{V.u = t.u\}. \end{array} \tag{3.1}$$

We present the attribute rules for each production rule within curly braces. When a nonterminal symbol appears multiple times in a production rule, we differentiate between its instances in the associated attribute rules by enumerating them. For instance, in the attribute rule of the first production rule in Equation (3.1), $P1$ refers to the first occurrence of P (the symbol on the left-hand side of the production rule) and $P2$ refers to the second occurrence (the P in the right-hand side).

Using the grammar presented in Equation (3.1), we can derive the expression for accelerated motion, $c * a * t * t$, as shown in the parse tree depicted in Figure 3.1. The measurement unit (attribute value) of each nonterminal symbol is indicated in subscript.

Probabilistic context-free grammars (PCFGs) that encode dimensional consistency directly exist but tend to become unwieldy and large, making it challenging to incorporate other domain-specific knowledge into the production rules of the grammar. By instead encoding the rules for dimensionally-consistent arithmetic in attribute rules, we preserve the elegance and interpretability of CFGs. This allows us to easily combine dimensional consistency, captured in attribute rules, with other types of domain knowledge expressed in production rules.

3.4 From PAG to PCFG

We utilize grammars as a means of generating candidate expressions for equation discovery. Although attribute grammars provide an elegant and concise representation of the search space for expressions, there is no straightforward method for randomly sampling expressions from an attribute grammar. However, it is feasible to convert a dimensional attribute grammar into a context-free grammar by creating new nonterminal symbols that

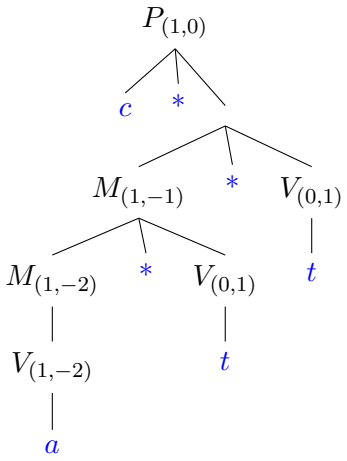


Figure 3.1: Parse tree for the equation $x = a * t^2$, derived with the attribute grammar in Equation (3.1). The blue color indicates terminal symbols, while the black color stands for nonterminal symbols.

Algorithm 3.1: TRANSFORM_GRAMMAR($\mathcal{G}$,$\mathcal{U}$)
Transform a probabilistic attribute grammar to a dimensionally-consistent PCFG.

Data: attribute dimensional grammar: $\mathcal{T},\ \mathcal{N},\ \mathcal{R},\ \mathcal{U},\ S \in \mathcal{N}$
Result: dimensionally-consistent PCFG: $\mathcal{T},\ \mathcal{N}',\ \mathcal{R}',\ S' \in \mathcal{N}'$

initialize $\mathcal{R}' = \{\},\ \ \mathcal{N}' = \{\},\ \ \mathcal{T}' = \mathcal{T}$;
for *production rule* $r \in \mathcal{R},\ r = A_0 \rightarrow A_1 A_2 \dots A_n [p]\{\kappa\}$ **do**
  initialize $\mathcal{R}'_r = \{\}$;
  for *each* $s = (\langle A_0, u_0 \rangle, \langle A_1, u_1 \rangle, \dots, \langle A_n, u_n \rangle) \in (\mathcal{N} \times \mathcal{U})^*$ **do**
    if *check*(κ, s) **then**
      $\mathcal{N}' = \mathcal{N}' \cup \{\langle A, u \rangle \in s\}$;
      $\mathcal{R}'_r = \mathcal{R}'_r \cup \{\langle A_0, u_0 \rangle \rightarrow \langle A_1, u_1 \rangle \dots \langle A_n, u_n \rangle\ [p]\}$;
    end
  end
  for *each* $r' = \langle A_0, u_0 \rangle \rightarrow \langle A_1, u_1 \rangle \dots \langle A_n, u_n \rangle\ [p],\ r' \in \mathcal{R}'_r$ **do**
    $p(r') = p(r)\ /\ |\{\langle A_0, u_0 \rangle \rightarrow \alpha\ \in\ \mathcal{R}'_r\}|$;
  end
  $\mathcal{R}' = \mathcal{R}' \cup \mathcal{R}'_r$;
end

enumerate all possible attribute values and then expanding the set of production rules while adhering to the constraints specified by attribute rules.

The procedure for converting a dimensional attribute grammar to a probabilistic context-free grammar is outlined in Algorithm 3.1. This approach involves constraining the possible attribute values to a finite set of units denoted by $\mathcal{U}$. Consider the attribute grammar given in Equation (3.1), with the starting symbol $S = P,\ P.u = (1, 0)$, and the unit set $\mathcal{U} = m, s, ms^{-1}, ms^{-2}$.

To transform this attribute grammar, we first initialize the sets of nonterminal symbols $\mathcal{N}'$ and production rules $\mathcal{R}'$ of the PCFG to empty sets. For each production rule $r = A_0 \rightarrow A_1 A_2 \dots A_n\ [p]\kappa$ in the attribute grammar, where p is the probability and κ represents the attribute rules, we create a temporary set of production rules denoted by $\mathcal{R}'_r$.

Next, we iterate through all possible combinations of attribute value assignments to nonterminal symbols in production rule r. Specifically, we iterate through all possible sequences of nonterminal symbol-physical unit pairs $s = (\langle A_0, u_0\rangle \langle A_1, u_1\rangle \dots \langle A_n, u_n\rangle); \quad u_i \in \mathcal{U}$. For instance, for the production rule $M \to M * V[p_M] M1.u = M2.u + V.u$ in our example, we iterate through the sequences:

$$
\begin{aligned}
&(\langle M, (1,0)\rangle, \langle M, (1,0)\rangle, \langle V, (1,0)\rangle)\\
&(\langle M, (1,0)\rangle, \langle M, (1,0)\rangle, \langle V, (0,1)\rangle)\\
&(\langle M, (1,0)\rangle, \langle M, (1,0)\rangle, \langle V, (1,-1)\rangle)\\
&\dots\\
&(\langle M, (1,-2)\rangle, \langle M, (1,-2)\rangle, \langle V, (1,-2)\rangle).
\end{aligned}
$$

check(κ, s) is a function that evaluates whether the sequence s fulfills all the constraints defined in the attribute rule κ, which belong to production rule r. For example:

$$
\begin{aligned}
&\textit{check}\ (M1.u == M2.u + V.u,\ (\langle M, (1,0)\rangle, \langle M, (1,-1)\rangle, \langle V, (0,1)\rangle)\\
&= ((1,0) == (1,-1) + (0,1))\\
&= \checkmark.
\end{aligned}
$$

For a dimensional grammar, the implementation of *check*(κ, s) is a straightforward process since attribute rules solely involve physical units that are represented as vectors. Once the *check* function has been performed, we proceed to construct new nonterminal symbols from the symbol-unit pairings $\langle A_n, u_n\rangle \in s$ of each sequence that has passed the *check*(κ, s) test, and then add them to the set of PCFG nonterminals $\mathcal{N}'$. Similarly, we construct new production rules $\langle A_0, u_0\rangle \to \langle A_1, u_1\rangle \dots \langle A_n, u_n\rangle$ and add them to $\mathcal{R}'_r$. For the sake of simplicity and readability, we denote the pair $\langle A, u\rangle$ as A_u.

In our example, the aforementioned sequence successfully passes the *check*, and we consequently update $\mathcal{N}'$ and $\mathcal{R}'$.

$$
\begin{aligned}
\mathcal{N}' &= \mathcal{N}' \cup \{M_{(1,0)}, M_{(1,-1)}, V_{(0,1)}\}\\
\mathcal{R}'_r &= \mathcal{R}'_r \cup \{M_{(1,0)} \to M_{(1,-1)} * V_{(0,1)}\ [p_M]\}.
\end{aligned}
$$

After completing production rule r, we must normalize the probabilities of the newly created productions to ensure proper distributions. To achieve this, we iterate through the new production rules in the set $\mathcal{R}'_r$, and for each one, we assign it the probability of rule r, divided by the number of production rules in $\mathcal{R}'_r$ that share the same nonterminal on the left-hand side as the production rule in question. For the production rule $r' = M_{(1,0)} \to M_{(1,-1)} * V_{(0,1)}$ in our example:

$$
\begin{aligned}
&\{M_{(1,0)} \to \alpha\ \in\ \mathcal{R}'_{M \to M*V}\} = \left\{ \begin{array}{l} M_{(1,0)} \to M_{(1,-1)} * V_{(0,1)}, \\ M_{(1,0)} \to M_{(0,1)} * V_{(1,-1)} \end{array} \right\}\\
&p(r') = p(r)\ /\ |\{M_{(1,0)} \to \alpha\ \in\ \mathcal{R}'_{M \to M*V}\}| = p_M\ /\ 2.
\end{aligned}
$$

Finally, we add the set $\mathcal{R}'_r$ to the set $\mathcal{R}'$ and then proceed to the next production rule r in $\mathcal{R}$. Equation (3.2) contains the entire output of Algorithm 3.1 for the example grammar in Equation (3.1) with the unit set $\mathcal{U} = \{m, s, ms^{-1}, ms^{-2}\}$ and starting symbol $S =$

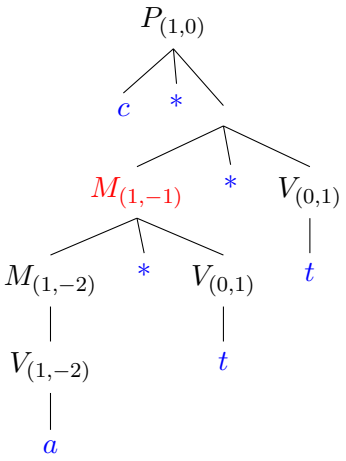


Figure 3.2: Parse tree for the equation $x = a * t^2$, derived with the attribute grammar in Equation (3.1). The blue color indicates terminal symbols, while the black color stands for nonterminal symbols. The parse tree cannot be derived by a PCFG version of the grammar using the minimal unit set $\mathcal{U} = \{m, s, ms^{-2}\}$, since we cannot compose the red nonterminal symbol $M_{(1,-1)}$ without the auxiliary unit $ms^{-1} = (1, -1)$.

$P,\ P.u = (1, 0)$:

$$
\begin{aligned}
P_{(1,0)} &\to P_{(1,0)} + c * M_{(1,0)}\ [p_P] \quad | \quad c * M_{(1,0)}\ [1 - p_P] \\
M_{(1,0)} &\to M_{(1,-1)} * V_{(0,1)}\ [p_M/2] \quad | \quad M_{(0,1)} * V_{(1,-1)}\ [p_M/2] \quad | \quad V_{(1,0)}\ [1 - p_M] \\
M_{(1,-1)} &\to M_{(1,-2)} * V_{(0,1)}\ [p_M/2] \quad | \quad M_{(0,1)} * V_{(1,-2)}\ [p_M/2] \quad | \quad V_{(1,-1)}\ [1 - p_M] \\
M_{(1,-2)} &\to V_{(1,-2)}\ [1.0] \\
M_{(0,1)} &\to V_{(0,1)}\ [1.0] \\
V_{(1,-2)} &\to a\ [1.0] \\
V_{(0,1)} &\to t\ [1.0].
\end{aligned}
\tag{3.2}
$$

3.5 The Unit Set and Auxiliary Units

The choice of the unit set $\mathcal{U}$ plays a crucial role in transforming an attribute grammar into a context-free grammar. The most straightforward and economical option for $\mathcal{U}$ is to use the set of units of measured quantities relevant to the problem at hand. However, note that the unit set utilized in the example presented in Equation (3.2) includes the unit of velocity, $(1, -1)$, which is not among the units of measured quantities. This unit is necessary to derive the expression $c*a*t*t$, as can be observed from the parse tree depicted in Figure 3.2. Since a PCFG instance of the attribute grammar in Equation (3.1) can only generate sums and differences of pairs of units from the finite set $\mathcal{U}$, it is impossible to derive the unit $(1, -2)$ without an intermediate step of either $(1, -1)$ or $(0, 2)$.

We call units such as $(1, -1)$ in the accelerated motion example *auxiliary* units, as they facilitate the derivation of expressions but do not have any corresponding terminal symbols. Deciding whether or not to include such auxiliary units in $\mathcal{U}$, and which to include, poses a nontrivial problem. In the case of $x = \frac{1}{2}at^2$, which deals with the kinetics problem, it is reasonable to assume that velocity is an essential quantity, in addition to acceleration and time. However, in some cases, relevant background knowledge may not be available.

Algorithm 3.2: EXTEND_UNITS $(U, \boldsymbol{u}_y)$
Extend the set of units with the required auxiliary units.

Data: unit matrix U, target variable unit: $\boldsymbol{u}_y$
Result: extended unit matrix U'

1 $\mathbf{k} = \mathit{solve\_diophantine\_equation}(\mathbf{u}_y = U \cdot \mathbf{k})$;
2 define $Z^{(n)} = \{i \cdot sgn(n) : 0 \leq i \leq |n|\}$;
3 set of expanded coefficients $\mathcal{H} = Z^{(k_1)} \times Z^{(k_2)} \times \cdots \times Z^{(k_d)}$;
4 matrix of extended coefficients $H = \begin{bmatrix} \boldsymbol{h}_1^T & \boldsymbol{h}_2^T & \dots \end{bmatrix} : \boldsymbol{h}_i \in \mathcal{H}$;
5 extended unit matrix $U' = U \cdot H$;

A possible universal approach to selecting auxiliary units would be to systematically enumerate all the units up to a certain maximum exponent (e.g., (-2,0), (-1,-1), (-1,0), (-1,1), (0,-2), (0,-1), (0,0), (0,1), (0,2), (1,-1), (1,0), (1,1), (2,0) for our example). However, this idea suffers from combinatorial explosion, where the number of added units increases steeply with the maximum order, as well as with the number of base units required. As a result, this approach leads to a large number of production rules and impairs the efficiency of the grammar.

Our heuristic procedure for extending the unit set of a mathematical grammar with auxiliary units is summarized in Algorithm 3.2. This method is applicable to any grammar that modifies units exclusively through multiplication and division of up to two quantities at a time. By using this procedure, we can introduce a sufficient set of units for a functioning PCFG while minimizing the number of auxiliary units needed, compared to the naive approach.

The first step of our procedure involves formulating and solving the diophantine equation $\boldsymbol{u}_y = U \cdot \boldsymbol{k}$, where $\boldsymbol{u}_y$ is the target variable unit, U is a matrix with units of observed quantities as columns, and $\boldsymbol{k}$ is a vector of integer coefficients that solve the equation. Next, we construct the coefficient matrix H, which includes all integer vectors that lie within or at the edge of the box spanned by $\boldsymbol{k}$ and the zero vector. Finally, we obtain the extended unit matrix U' by transforming the coefficient set H back into the unit space using $U' = U \cdot H$. To illustrate the procedure, we demonstrate its application using the well-known example of accelerated motion. The unit matrix is

$$U = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix},$$

with the first column corresponding to acceleration (ms^{-2}) and the second to time (s). To obtain a dimensionally consistent equation with $(1, 0)$ on the left-hand side, we must solve the diophantine equation

$$\begin{bmatrix} 1 & 0 \end{bmatrix} = U \cdot \boldsymbol{k} \quad \rightarrow \quad \boldsymbol{k} = \begin{bmatrix} 1 & 2 \end{bmatrix}.$$

Here, the left-hand side of the diophantine equation represents the target unit, which is the unit of the dependent variable x that we want to obtain. The solution $\boldsymbol{k} = (1, 2)$ demonstrates that the linear transformation $1 \cdot (1, -2) + 2 \cdot (0, 1)$ of the units of the variables on the right-hand side leads to the unit $(1, 0)$ that we need on the left-hand side. However, since the multiplication operator only allows for simple addition of units and not their multiplication by an integer constant, we must introduce all the intermediate linear combinations $a \cdot (-1, 2) + b \cdot (0, 1)$, where $0 \leq a \leq 1$ and $0 \leq b \leq 2$, to allow for step-by-step

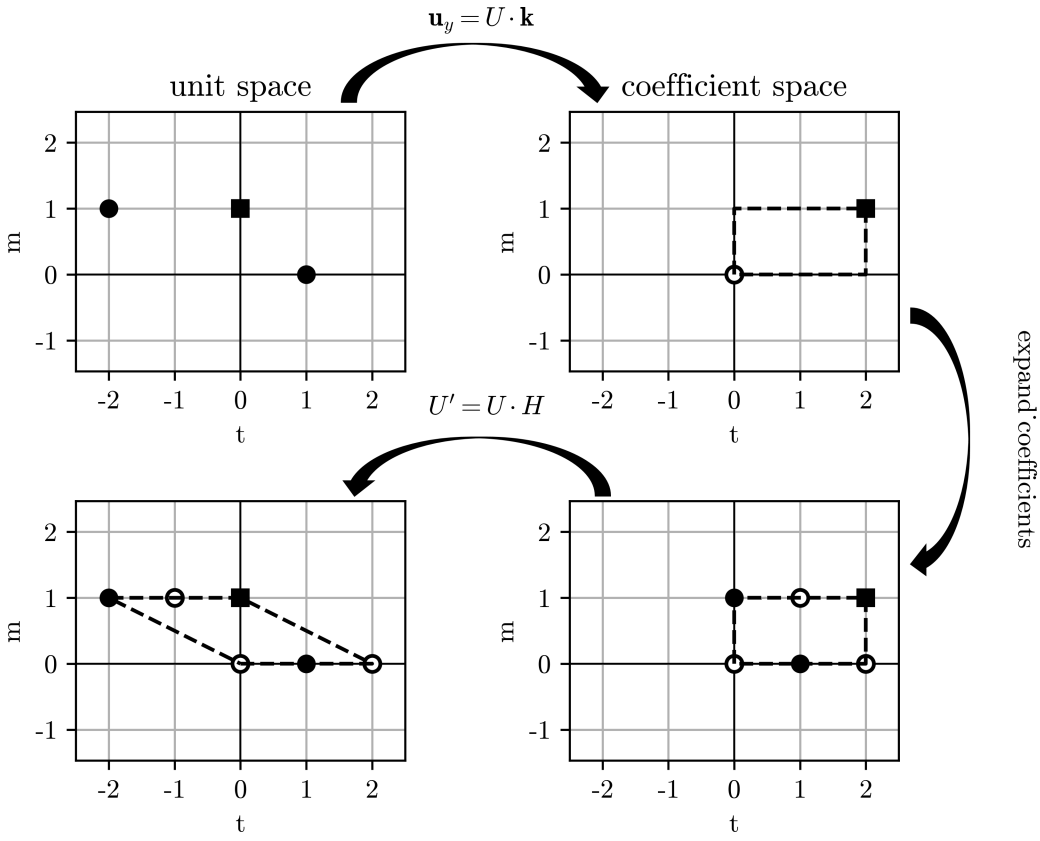


Figure 3.3: Graphical representation of the main steps of *expand_ units* (Algorithm 3.2, demonstrated on the example problem $x = at^2$; $\{u_x = m = (1,0), u_a = ms^{-2} = (1,-2), u_t = s = (0,1)$. The plots on the left-hand side are in the space of measurement units, while the plots on the right-hand side are in the space of solution coefficients. Square symbols correspond to the dependent variable unit u_x, full circles correspond to the units of independent variables u_a and u_t and empty circles represent the auxiliary units added by *extend_ units*. The dashed lines represent the box spanned by 0 and the solution coefficients. Added units are within, or at the border of the box.

derivation of the goal unit $(1,0)$. Using the solution $\boldsymbol{k}$ we construct the matrix H as

$$\mathcal{H} = Z^{(k_1)} \times Z^{(k_2)} = \{0,1\} \times \{0,1,2\} \quad \rightarrow \quad H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 & 1 & 2 \end{bmatrix}.$$

Finally, we obtain the extended unit matrix:

$$U' = U \cdot H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 2 & -2 & -1 & 0 \end{bmatrix}.$$

We observe that the *extend_ units* procedure has successfully extended the unit set with the auxiliary units $(1,-1)$ and $(0,2)$ (in addition to the always-included dimensionless $(0,0)$ unit). As mentioned earlier, the addition of either of these units is sufficient to enable the grammar to generate the correct solution. Among the two auxiliary units, only $(1,-1)$ has physical significance as the unit of velocity, whereas the unit $(0,2)$ is unnecessary. Nonetheless, the overall number of auxiliary units added with the *extend_ units* method remains acceptably low. For a visual representation of this example, please refer to Figure 3.3.

Table 3.1: Number of parse trees with height up to and including h derived by the polynomial PCFG and its dimensionally-consistent counterpart, constructed for the task of discovering the expression at^2 (Eqs. (2.24) and (3.2)). The lowest height possible with the unrestricted grammar is $h = 3$, corresponding to the expressions $c \cdot a$ and $c \cdot t$. On the other hand, the lowest height possible with the dimensional grammar is 5. The dimensional grammar derives two different parse trees with height 5, both of which correspond to the expression $c \cdot a \cdot t \cdot t$.

grammar / h	3	4	5	6	7	8	9	10
unrestricted	2	18	266	$8 \cdot 10^3$	$4 \cdot 10^5$	$6 \cdot 10^7$	$1 \cdot 10^{10}$	$8 \cdot 10^{12}$
dimensional	0	0	2	6	14	30	62	126

3.6 Effect on the Search Space Size

The main goal of ensuring dimensional consistency in equation discovery is to decrease the size of the search space, which enables us to examine a smaller number of candidate expressions. To demonstrate the significant impact that dimensional consistency can have, we can compare the number of parse trees generated by an unconstrained grammar and its dimensionally-consistent version [73]. In Table 3.1, we present a comparison between the number of parse trees with a maximum height of h, generated by the polynomial grammar defined in Equation (2.24), and the number of parse trees generated by its dimensionally-consistent counterpart. When considering the dimensionally-consistent grammar, the number of parse trees increases much less drastically as the parse tree height increases.

3.7 Random Expression Generation

We employ probabilistic context-free grammars (PCFGs) as generators of random mathematical expressions, which serve as candidate models in the equation discovery process. The sampling process [73] initiates with the starting nonterminal symbol. It is worth noting that in the process of transforming the attribute grammar into a PCFG, each nonterminal symbol is merged with a physical unit. The starting symbol is joined to the target variable unit. The procedure proceeds recursively, replacing nonterminal symbols in the current string with combinations of nonterminal and terminal symbols by following the production rules, until only terminal symbols remain. At each step, the procedure selects a production rule at random from all the rules that have the selected nonterminal symbol on the left-hand side.

The presence of dead ends – nonterminal symbols without corresponding production rules – is a complication introduced by dimensionally-aware PCFGs in the sampling process. We tackle the issue of dead ends by employing a form of acceptance-rejection sampling. Whenever the sampling process encounters a dead end, we simply restart the entire process with a different random seed. The number of auxiliary units in a grammar determines the number of dead ends and, therefore, the number of restarts during sampling.

This solution can considerably increase the time required to generate a batch of candidate expressions. However, the primary bottleneck in generate-and-test approaches to equation discovery is parameter estimation. As long as the number of dead ends in a grammar is not excessive, the time taken to generate a candidate expression is negligible compared to the time required to evaluate it.

3.8 Empirical Analysis

To showcase the effectiveness of our dimensional grammar approach and assess its performance, we return to the Feynman database for symbolic regression we introduced in Section 2.4.1. This time, we compare the performance of an unrestricted universal grammar with its dimensionally-consistent counterpart.

3.8.1 Experimental setup

For each problem from the Feynman database, we construct a dimensionally-consistent universal mathematical grammar:

$$
\begin{array}{lcl}
E & \rightarrow & E + F \ [0.2] \ \{E1.u = E2.u = F.u\} \\
 & \rightarrow & E - F \ [0.2] \ \{E1.u = E2.u = F.u\} \\
 & \rightarrow & F \ [0.6] \ \{E.u = F.u\} \\
F & \rightarrow & F * T \ [0.2] \ \{F1.u = F2.u + T.u\} \\
 & \rightarrow & F \ / \ T \ [0.2] \ \{F1.u = F2.u - T.u\} \\
 & \rightarrow & T \ [0.6] \ \{F.u = T.u\} \\
T & \rightarrow & (E) \ [0.12] \ \{T.u = E.u\} \\
 & \rightarrow & F(E) \ [0.08] \ \{T.u = E.u, \ T.u = (0,0,0,0,0)\} \\
 & \rightarrow & V \ [0.4] \ \{T.u = V.u\} \\
 & \rightarrow & c \ [0.4] \ \{T.u = (0,0,0,0,0)\} \\
F & \rightarrow & f_1 \ [1/11] \ \{\} \mid \ \ldots \ \mid f_{11} \ [1/11] \ \{\} \\
V & \rightarrow & q_1 \ [1/m] \ \{V.u = q_1.u\} \mid \ \ldots \ \mid q_m \ [1/m] \ \{V.u = q_m.u\},
\end{array}
\tag{3.3}
$$

By utilizing the universal grammar, it is possible to generate any mathematical expression that is constructed using the four fundamental operations (+,-,*,/) along with a selection of transcendental functions and the square root operation. To achieve this, we make use of Algorithm 3.1 to convert the attribute grammar into a Probabilistic Context-Free Grammar (PCFG), while Algorithm 3.2 is employed to guarantee an adequate set of units.

For each problem in the Feynman database, we generate $3 \cdot 10^4$ distinct candidate expressions randomly using two approaches: the dimensionally-consistent universal mathematical PCFG and the unrestricted version of the universal mathematical PCFG (i.e., the grammar in (3.3) without attribute rules). It is worth noting that for certain problems, the space of candidate equations is restricted to such an extent by dimensional consistency that generating $3 \cdot 10^4$ unique expressions is impossible. This is, in fact, a desirable outcome since the estimation of candidate expression parameters consumes a significant amount of the computational resources utilized by ProGED.

To further minimize the computational time, we exclude expressions that contain more than five numerical constants. We fit the parameters of each candidate expression to the data and estimate its goodness-of-fit by calculating the relative root mean squared error (ReMSE). We then rank the expressions based on their ReMSE values. When evaluating our approach, our main concern is whether it can successfully solve a given problem, as well as the amount of computational resources required to solve it, on average. We consider a problem to be solved successfully if the error of the best candidate equation is below a specified threshold. As the data is devoid of noise, we impose a strict threshold of ReMSE $< 10^{-9}$. Additionally, we manually examine the expressions that surpass the threshold and ensure that they match the ground truth, which is the original equation from the Feynman database.

Table 3.2: The number of successfully reconstructed equations from the Feynman database (out of 100), comparing ProGED using the unrestricted universal grammar, ProGED using the dimensionally-consistent universal grammar and Deep Symbolic Optimization (DSO) [14]. All three methods were limited to evaluating at most 30000 candidate equations.
[a]We ran DSO with random seeds $0, 1, 2$ and 3, resulting in $54, 51, 52$ and 54 reconstructed equations, respectively.

method	# reconstructed eqns
ProGED – unrestricted grammar	36
ProGED – dimensional grammar	58
Deep Symbolic Optimization	$51 - 54^a$

3.8.2 Deep symbolic optimization

To evaluate the performance of our approach in comparison to other methods in the field, we conduct a comparison with DSO (Deep Symbolic Optimization) [14], using the implementation provided in its public repository. In order to ensure a fair comparison, we constrain DSO to the same number of samples (parameter estimations) as ProGED, which is $3 \cdot 10^4$, and allow it to perform parameter estimation. We set the minimum number of tokens to 3 and the maximum to 128, limit the number of numerical constants to five (same as ProGED), and include the transcendental functions required for the problems in the Feynman database. We utilize all implemented background knowledge priors with default settings. Although the literature mentions dimensional constraints [36], they do not seem to be integrated into the public repository. Consequently, we were unable to include a dimensionally-consistent version of DSO in our comparison. We conduct four DSO runs, each with a different random seed (0-3).

3.8.3 Results

The results of our empirical analysis are summarized in Table 3.2 and detailed in Appendix C. Our use of a dimensionally-consistent universal grammar allowed for the successful reconstruction of 58 out of 100 equations from the Feynman database, which is a significant improvement over the 36 equations that were reconstructed using our previous approach that relied on an unrestricted universal grammar.

When allowed the same number of samples as ProGED, DSO was able to reconstruct between 51 and 54 equations from the Feynman database. For completeness, we also tested DSO with the default $2 \cdot 10^6$ samples, both with and without numerical constants (which was computationally very demanding). In the former case, DSO solved 39 problems from the Feynman database, and in the latter case, it solved 83.

These results demonstrate that dimensional consistency has a significant impact on the performance of ProGED, making it comparable to DSO, a powerful deep learning method. However, it is worth noting that DSO was able to discover up to 54 equations without relying on knowledge of physical units, which highlights the strength of the reinforcement learning approach. It is reasonable to assume that the performance of DSO would be even further improved by incorporating dimensional consistency.

In the rest of this chapter, we delve deeper into the analysis of ProGED's results. By examining both successful and unsuccessful reconstruction attempts, we can identify several interesting groups of problems from the Feynman dataset, which are summarized in Table 3.3.

Out of the 58 problems that were solved using the dimensionally-consistent grammar,

Table 3.3: Properties of interesting categories of problems from the Feynman dataset, grouped through manual inspection of experimental results. Columns from left to right: 1) "yes" if the problems in the group were successfully reconstructed using the dimensionally-consistent grammar, 2) number of problems in the group, 3) mean number of variables among the tasks in the group, 3) mean string length as a measure of complexity in the group, 4) mean number of unique candidate expressions in the group. Rows, from top to bottom: 1) problems that are easy with or without dimensions, 2) problems that are significantly easier with dimensional consistency, 3) problems that were solved thanks to dimensional consistency, 4) problems that were too difficult for our approach, 5) problems for which dimensional consistency introduced issues, 6) problems which dimensional analysis cannot help solve.

Group / property	**suc.**	**#prob.**	**#var.**	**compl.**	**#unique**
easy	yes	14	2.7	12	21k
easier w/ dim.	yes	20	2.8	13	500
possible due to dim.	yes	24	3.6	23	16k
dim. issues	no	5	3.2	26	12k
dim. can't help	no	6	2.8	35	20k
too complex	no	32	4.8	35	15k

34 were also solved using the unconstrained grammar. These problems were generally easier, with lower mean expression complexity and a smaller mean number of variables than the other groups. Fourteen of these problems represented a similar level of challenge for both grammars and involved equations such as

$$F = \mu N;\ u_F = u_N = kgms^{-2}, u_\mu = 1.$$

The other 20 problems formed the second group and were more challenging for the dimensionally-consistent grammar, resulting in significantly fewer unique candidate expressions being generated, on average 500 compared to 21,000 in the first group. Generating fewer candidate expressions means less computational effort is required for the testing step of equation discovery, which is the most computationally expensive step. These differences between the two groups demonstrate the efficiency improvements achieved by constraining the search space using dimensionally-consistent grammars. An example equation from this group is

$$L = mrv\sin\theta;\ u_L = kgm^2s^{-1}, u_m = kg, u_r = m, u_v = ms^{-1}, u_\theta = 1.$$

It is important to note that despite the improvements achieved by using dimensional analysis, there were still equations that could not be reconstructed using our approach. Four of the equations were impossible to express using a grammar that only allows dimensionless arguments to special functions, as they featured a square root of a dimensioned term, for example:

$$c = \sqrt{\gamma p/\rho};\ \{u_c = ms^{-1}, u_\gamma = 1, u_p = kgms^{-1}, u_\rho = kgm^{-3}\}.$$

Two of the problematic equations were solved by the unconstrained grammar, indicating that there are cases where an unrestricted grammar can be more effective than a dimensionally-aware one. Six of the problematic equations featured only dimensionless quantities, meaning that the dimensionally-aware grammar and the unrestricted grammar are equivalent in these cases. One such example is

$$f = e^{-(\theta/\sigma)^2/2}/\sqrt{2\pi\sigma};\ \{u_f = u_\theta = u_\sigma = 1\}.$$

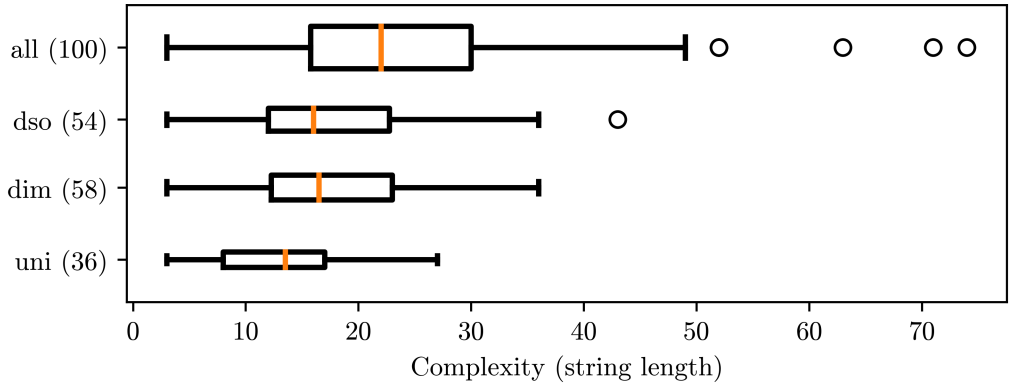


Figure 3.4: Comparison of the expression complexity of problems from the Feynman database, solved by the unrestricted universal grammar (uni), the dimensionally-consistent universal grammar (dim) and the complexity of all the problems in the database (all). The length of the string representation of the target mathematical expression serves as a measure of problem complexity. The median of each distribution is represented by an orange bar. The number of examples in each group is given in brackets before the name of the group and is proportional to the width of each box plot. Circles represent outlier examples in a distribution.

For such problems, dimensional analysis is not a useful tool, and using a dimensionally-aware grammar offers no improvement. The remaining 32 problematic equations were simply too complex to discover in the allotted computation time using our approach, such as

$$L = h/(2\pi)\omega^3/(\pi^2 c^2(e^{(h/(2\pi))\omega/(k_B T)} - 1));$$

$$\{u_L = kgs^{-2}, u_h = kgm^2s^{-1}, u_c = ms^{-1}, u_\omega = s^{-1}, u_{k_B} = kgm^2s^{-2}K^{-1}, u_T = K\}.$$

These equations had the highest mean number of variables and mean expression complexity among all groups, highlighting the limitations of the equation discovery approach based on Monte-Carlo sampling. Overall, the success of our approach in reconstructing equations from the Feynman database demonstrates the power of combining machine learning with dimensional analysis. However, there is still room for improvement, and more sophisticated solutions are needed to tackle complex problems that cannot be solved using current methods. The expression complexity distributions of problems reconstructed using the dimensionally-consistent grammar and the unrestricted grammar, as well as the distribution of expression complexity across the entire Feynman database, are shown in Figure 3.4. Our analysis reveals that the dimensionally-consistent grammar was capable of reconstructing more complex expressions than the unrestricted grammar, covering a significant portion of the complexity-space of the database. However, the tails of the distribution of expression complexity of all problems in the Feynman database remain beyond the reach of our methods.

Dimensionally-consistent grammars offer a means of constraining the search space for mathematical expressions, enabling the discovery of equations that would otherwise be too complex to reconstruct. Moreover, searching within the constrained space is more efficient, as it requires generating and evaluating fewer candidate expressions. To investigate the efficiency gains further, we perform bootstrapped resampling for each problem from the Feynman database and approximate a performance curve using the resulting data.

After running the equation discovery algorithm, we obtain a list of N candidate equa-

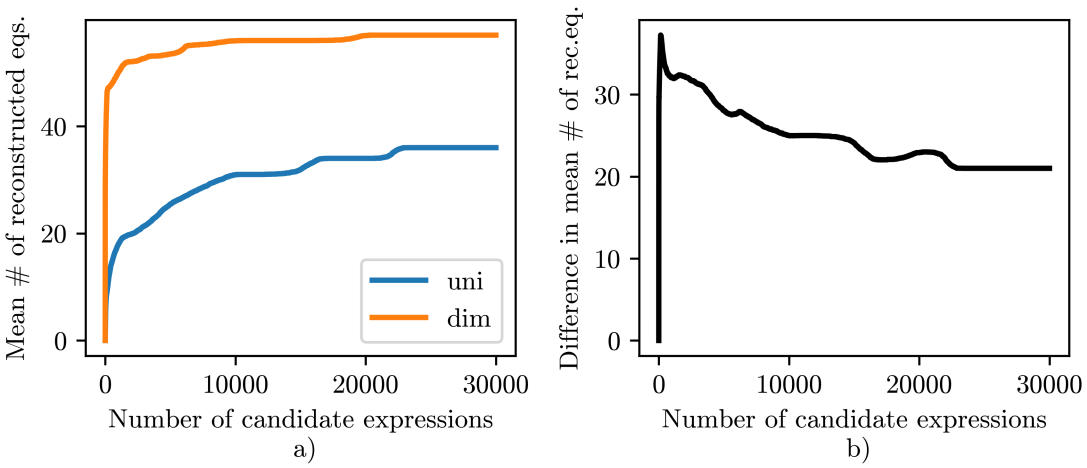


Figure 3.5: a) comparison of approximate performance curves of equation discovery using a universal mathematical PCFG (blue) and a dimensional version of the PCFG (orange) on the Feynman symbolic regression database. The horizontal axis depicts the number of sampled candidate expressions, while the vertical axis represents the number of reconstructed equations (out of 100), averaged across 1000 bootstrap samples. b) The difference between the approximate performance curves for the two grammars.

tions, each accompanied by an error-of-fit value and a probability of the right-hand side expression derived from the grammar. We then randomly sample a sequence of N models from this list, selecting each without repetition and weighting them by their respective probabilities. Using this sequence, we calculate the cumulative minimum of the error-of-fit values and generate a single performance curve. This process is repeated 1000 times with different random seeds, and the resulting learning curves are averaged. The resulting curve provides an estimate of the expected best error-of-fit that the algorithm would achieve for a given number of models sampled, simulating the equation discovery experiment multiple times.

The left-hand side of Figure 3.5 displays the approximate performance curves for the dimensionally-consistent universal grammar and the unrestricted universal grammar. Our analysis shows that the dimensionally-consistent grammar is capable of reconstructing over 40 equations from the Feynman dataset while requiring less than 1000 candidate expressions for each problem. This indicates that the dimensionally-consistent grammar is able to discover more equations while generating fewer expressions from the grammar, highlighting its significantly higher efficiency compared to the unrestricted grammar. On the right-hand side of Figure 3.5, we show the curve obtained by subtracting the approximate performance curve of the unrestricted grammar from that of the dimensionally-consistent grammar. The difference is most pronounced for small numbers of sampled expressions and decreases as more expressions are sampled, ultimately leading to the convergence of the two curves. This suggests that dimensional consistency offers the most significant benefits for resource-limited scenarios, which are the most common in practical use cases. These findings are promising and have important implications for the use of dimensionally-consistent grammars in real-world applications.

Chapter 4

Probabilistic Attribute Grammars

So far we have laid the groundwork for equation discovery using probabilistic context-free grammars and introduced the concept of probabilistic attribute grammars for ensuring dimensional consistency in generated equations. However, the approach in Chapter 3 relies on transforming a PAG into a PCFG by enumerating attribute values. Since a PCFG is limited to a finite (and relatively small) number of nonterminal symbols, this transformation is viable only for limited types of attributes and their associated rules. Furthermore, the question of which attribute values to include is nontrivial, as evident from our efforts to identify the required auxiliary units in Chapter 3.

In many instances of scientific and engineering applications, there is a wealth of context-specific knowledge available. This knowledge, however, often cannot be conveniently encoded in a context-free grammar. In order to be able to express the various types of more complex background knowledge, available for equation discovery, an algorithm for generating random expressions from a PAG is required that does not rely upon the transformation to a PCFG and can handle the expressive power and flexibility of probabilistic attribute grammars.

In this chapter, we develop a general purpose direct sampling algorithm for PAGs. As a demonstration of its utility, we present three specific examples, each illustrating the algorithm's applicability in generating equations with differing sets of attributes. The examples demonstrate the use of measurement units to ensure dimensional consistency, the encoding of particular properties of coupling terms in dynamical systems and an expression of Kirchoff's laws to guide the generation of equations governing electronic circuits.

4.1 Rethinking Probabilistic Attribute Grammars

In Chapter 3, we defined (loosely following [72]) an attribute grammar as an extension of probabilistic context-free grammars that allows for the specification of attributes for each (nonterminal or terminal) symbol in the grammar. Attribute values can be defined through attribute rules associated with the production rules of the grammar, and can be used to express relationships between attributes of different symbols, as well as constraints on attribute values.

4.1.1 On attributes

We distinguish between two types of attributes based on how attribute values are propagated through the derivation (parse) tree.

- The values of the synthesized attributes are propagated bottom-up. The value of the synthesized attribute of the nonterminal A on the left-hand side of the production

rule $A \to \alpha$ is calculated from the values of the attributes of the symbols on the right-hand side α.

- The values of the inherited attributes are propagated top-down. The value of the inherited attribute of a symbol X on the right-hand side α of the production rule $A \to \alpha$ is calculated from the values of the attributes of A and the attributes of the other symbols in α.

Recall the grammars in Equations (3.1) and (3.3) from Chapter 3. There, the dimensional unit u is a synthesized attribute: its values are propagated through the derivation tree from Figure 3.1 from the leaves (variables with known units) to the start symbol in the root, representing the derived expression's unit. The two grammars do not use any other attributes.

4.1.2 On attribute rules

Attribute rules are statements associated with each production rule in the grammar and enable the calculation and verification of attribute values. We categorize the attribute rules according to two classification schemes. The first scheme introduces two groups of rules based on when they are being applied in the derivation process:

- Pre-selection rules are applied before selecting a particular production rule. In this way, we can choose production rules to be used in the derivation based on the values of the attributes computed by these attribute rules.
- Post-selection rules are applied after choosing a particular production rule.

The second scheme clusters attribute rules with respect to their role in the derivation:

- Assignments calculate the values of attributes.
- Conditions are logical statements involving the values of attributes. They can be used to constrain the selection of a production rule or to check the validity of the derivation.

The product of these two classification schemes is a set of four types of attribute rules. We present the attribute rules for each production rule within curly braces in the following order: pre-selection assignment-type rule, pre-selection condition-type rule, post-selection assignment-type rule and post-selection condition-type rule. Similarly to the convention introduced in Chapter 3, when a nonterminal symbol appears multiple times in a production rule, we differentiate between its instances in the associated attribute rules by enumerating them.

In terms of implementation details, an attribute can be any Python object, and an attribute rule is a string containing Python code. Assignment-type rules are computed using Python's *exec* function, whereas condition-type rules are evaluated using the function *eval*. Assignment-type rules can be composed of any number of individual statements, separated by semicolons. Condition-type rules can be composed of any number of logical statements, joined by logical operators. The example below uses inherited attributes in a polynomial grammar to ensure only even powers up to 10:

$$\begin{aligned} P &\to P + c * M \ [p_P] \ \{\text{"M1.d} = 1\text{"}, , \} \\ P &\to c * M \ [1 - p_P] \ \{\text{"M1.d} = 1\text{"}, , \} \\ M &\to M * x \ [p_M] \ \{\text{"M2.d} = \text{M1.d} + 1\text{"}, \text{"M1.d} < 10\text{"}, , \} \\ M &\to x \ [1 - p_M] \ \{, \text{"M1.u} \ \% \ 2 == 0\text{"}, , \}. \end{aligned} \tag{4.1}$$

In this example, only pre-selection attribute rules are used. Assignment-type rules are highlighted in blue and condition-type rules in olive color. The lone attribute is d – an inherited attribute that acts as a counter for recursion and represents the degree of the monomial. The two condition-type rules for the nonterminal M impose constraints on the two possible production rules for M – the first production (which recursively increases the power) can be chosen only while d is less than 11, and the second production (which ends the recursion) can be chosen only for even values of d. In this grammar, information only flows down the parse tree from the root node to the leaf nodes, where it is used to constrain the selection of production rules.

The next example uses synthesized attributes in a polynomial grammar to ensure that each expression contains the same number of appearances of the variables x and y:

$$
\begin{array}{lcl}
S & \rightarrow & P\ [1.0]\ \{,,,\text{"P1.x == P1.y"}\} \\
P & \rightarrow & P + c * M\ [p_P]\ \{,,\text{"P1.x = P2.x+M1.x; P1.y = P2.y+M1.y"},\} \\
P & \rightarrow & c * M\ [1 - p_P]\ \{,,\text{"P1.x = M1.x; P1.y = M1.y"},\} \\
M & \rightarrow & M * V\ [p_M]\ \{,,\text{"M1.x = M2.x+V1.x; M1.y = M2.y+V1.y"},\} \\
M & \rightarrow & V\ [1 - p_M]\ \{,,\text{"M1.x = V1.x; M1.y = V1.y"},\} \\
V & \rightarrow & 'x'\ [p_x]\ \{,,\text{"V1.x = 0; V1.y = 1"},\} \\
V & \rightarrow & 'y'\ [1 - p_x]\ \{,,\text{"V1.x = 1; V1.y = 0"},\}.
\end{array}
\tag{4.2}
$$

Here, only post-selection attribute rules are used. Assignment-type rules are highlighted in teal and condition-type rules in red. The synthesized attributes x and y act as counters for their respective variables. The production rule for the starting symbol S contains a condition that compares the values of the counter. If they do not match, the derivation is rejected and we repeat the random sampling. In this example, information only flows from leaf nodes of the parse tree up to the root node, where it is used to verify the derivation.

Our formulation of PAGs deviates from the established formulation by allowing for global attributes. For example, complex rules can be defined in external functions, which are added to the grammar's isolated namespace. A limited number of global structures is constructed by default: one for each nonterminal symbol in the grammar. These represent the nonterminal symbols in an abstract sense (and are unique), whereas local instances of nonterminals are created by each application of a production rule (and may create several copies of a given nonterminal). In production rules, local instances are enumerated, whereas their global counterparts are not. Among other benefits, global properties allow us to shorthand the passing of information upward. For instance, in the previous example, we could assign the attributes x and y to the global S and spare ourselves writing out all the assignment rules that merely pass information:

$$
\begin{array}{lcl}
S & \rightarrow & P\ [1.0]\ \{\text{"S.x=0; S.y=0"},,,\text{"S.x == S.y"}\} \\
P & \rightarrow & P + c * M\ [p_P]\ \{,,,\} \\
P & \rightarrow & c * M\ [1 - p_P]\ \{,,,\} \\
M & \rightarrow & M * V\ [p_M]\ \{,,,\} \\
M & \rightarrow & V\ [1 - p_M]\ \{,,,\} \\
V & \rightarrow & x\ [p_x]\ \{,\text{"S.x <= S.y"},\text{"S.x += 1"},\} \\
V & \rightarrow & y\ [1 - p_x]\ \{,\text{"S.x <= S.y"},\text{"S.y += 1"},\}.
\end{array}
\tag{4.3}
$$

Besides demonstrating the use of the global S to shorthand the passing of information upwards, we have also improved the grammar in this example by adding a pre-selection condition to each production rule for V, resulting in fewer derivation rejections. Interestingly, in this version of the grammar, p_x has no influence on the derivation process

anymore, since the attribute rules for V ensure an equal number of both variables in the derivation.

It must be noted that in our formulation, production rules are immutable. This means that attribute rules cannot directly affect the expression that is being derived, nor the production probabilities. Their influence on the derivation is limited to pre-selection condition rules constraining the selection of production rules and to post-selection condition rules accepting or rejecting the entire derivation. While this limits the power of attribute rules, it ensures that any expression generated by a PAG could also be generated by the PCFG obtained by stripping the PAG of attribute rules. Such a PCFG can also be used to parse any expression generated by the PAG.

4.2 Direct Sampling Algorithm

Having redefined our conceptualization of PAGs, we can introduce the direct sampling algorithm for PAGs. As an extension of Algorithm 1, it follows the same basic procedure. The algorithm begins with the starting symbol and recursively replaces nonterminal symbols with terminal and nonterminal symbols, until only terminal symbols remain. Whenever more than one production rule applies, one is chosen randomly according to their respective probabilities.

The chief novelty in the PAG version is that instead of randomly sampling from all production rules with the appropriate symbol on the left-hand side, the method first checks which production rules pass the pre-selection attribute rules. This consists of first executing assignment rules, then evaluating condition rules. The result is a subset of applicable production rules. The algorithm then chooses a production rule from this subset according to their (renormalized) probabilities.

After recursively expanding each nonterminal symbol on the right-hand side (just like in Algorithm 1), the method checks the post-selection attribute rules. The algorithm first executes the assignment rules, then evaluates condition rules. If the post-selection condition result is negative, the derivation is rejected. We present the entire procedure in Algorithm 4.1.

The generation of a single random expression can fail in two ways, each corresponding to one type of condition-type attribute rule. The first scenario (line 14 of Algorithm 4.1) represents the situation where none of the production rules with the appropriate symbol on the left-hand side passes the pre-selection condition attribute rule. Since there are no applicable production rules to sample from, we abort the derivation. The second scenario (line 37 of Algorithm 4.1 is a negative result of evaluating the post-selection condition-type attribute rule, in which case we also abort the derivation. Similarly to the procedure in Chapter 3, we repeat the random sampling until we successfully produce an expression. This can be considered a form of acceptance-rejection sampling [74].

The direct sampling algorithm also features a pair of default attributes, *frozen* and *subtree*, with behavior that deviates from the formulation in the previous section, since they directly affect the sampling procedure. The two attributes can be assigned to global nonterminals and enable the freezing of a particular nonterminal derivation. A nonterminal can be “frozen" by setting its *frozen* attribute to true in any assignment-type attribute rule. The first time this happens, the subtree it derives is stored in its *subtree* attribute. Henceforth, whenever this nonterminal is encountered in this sampling, the stored subtree is returned (line 3 of Algorithm 4.1) instead of the usual random sampling procedure. This functionality allows a grammar to generate a random term or sub-expression and then reuse it in its exact form elsewhere in the expression. For instance, this is important for generating coupling terms in dynamical systems – terms that appear in several coupled

Algorithm 4.1: GENERATE_SAMPLE_ATTRIBUTED($\mathcal{G}$, A)
Generate a random expression from a probabilistic attribute grammar.

Data: start symbol: $A \in \mathcal{N}$; attribute dimensional grammar: $\mathcal{T}$, $\mathcal{N}$, $\mathcal{R}$, $\mathcal{U}$, $S \in \mathcal{N}$
Result: expression, corresponding to randomly sampled parse tree, following the attribute rules

if this nonterminal has been frozen already, simply return its subtree;
if *A.frozen* **then**
  return *A.subtree*;
end
check which production rules pass the pre-selection condition attribute rules;
initialize valid_production_rules = [ ];
for *production_rule in production_rules* **do**
  execute(production_rule.pre_selection_assignment_rules);
  if *check(production_rule.pre_selection_condition_rules)* **then**
    valid_production_rules.append(production_rule);
  end
end
if *length(valid_production_rules) == 0* **then**
  raise DeadEndError;
end
Choose a random rule among valid_production_rules
$(A \to \alpha) \in \mathcal{R} : \alpha = A_1 A_2 \dots A_k$, $A_i \in \mathcal{N} \cup \mathcal{T}$;
go through the symbols on the left-hand side one by one, starting recursive calls for nonterminals;
initialize $(s, p) = ([\,], 1)$;
for $i = 1$, $i \leq k$ **do**
  if $A_i \in \mathcal{T}$ **then**
    $s = s$.append(A_i);
  else
    (s_i, p_i) = GENERATE_SAMPLE_ATTRIBUTED(G, A_i);
    $s = s$.append(s_i);
    $p = p \cdot p_i$;
  end
end
evaluate the post-selection attribute rules: if negative, reject this derivation;
execute(production_rule.post_recursion_assignment_rules);
if *check(production_rule.post_recursion_condition_rules,)* **then**
  if the nonterminal has been frozen in this call, store its subtree;
  if *A.frozen* **then**
    A.subtree = (s, p);
  end
  return (s, p);
else
  raise ConditionError;
end

equations.

The direct sampling algorithm enables the use of probabilistic attribute grammars as a powerful, general purpose framework for expressing and combining various types of background knowledge in equation discovery. In the remainder of this chapter we demonstrate its usage and flexibility by developing PAGs for different types of background knowledge in three domains.

4.3 Search Space Constriction

The main advantage PAGs have over PCFGs in equation discovery is a more constrained space of equations that needs to be searched. In order to get a better understanding of this difference, we developed a method for visualizing the many derivations, possible in a grammar, called aggregated parse trees (APTs).

To construct an APT, we first generate a large number of random expressions using a grammar and initialize a directed graph, then process the parse tree of each expression with the following recursive procedure. The root node in the graph corresponds to the starting symbol of the parse tree. For each non-terminal child, we create a unique identifier – the sequence of nonterminals defining the path from the starting symbol to the current symbol. If no edge exists between the parent and the new child node, we add one. Importantly, we keep count of how often each node and edge appears in the parse trees. The function operates recursively, running itself on each non-terminal child with that child becoming the new parent. The process continues until all paths in the parse tree have been explored and all non-terminal nodes and edges added to the graph.

When the function is applied to all parse trees generated from a certain grammar, the output graph is an aggregated parse tree containing all possible derivation paths from the set of parse trees. In the visualization, we depict each nonterminal in its own color. Furthermore, we visualize the frequency of each node and edge in the collection of parse trees using the transparency in the graph plot.

APTs are similar to aggregated expression trees (AETs), but have a different interpretation. Instead of aggregating expressions trees, APTs aggregate parse trees. This makes an APT specific to a grammar, unlike AETs, which can generalize any type of expressions generator. As such, directly comparing APTs of different grammars does not make sense. The nodes in an APT correspond to terminal and nonterminal symbols of the grammar and the edges correspond to production rules. Where an AET depicts the search space of mathematical expressions, an APT depicts the most likely pathways, taken during the random generation of samples from a grammar. We use APTs in this chapter, as they are very handy for visualizing constraints that attribute rules impose on derivations. Since a PAG and the PCFG obtained by stripping it of its attribute rules share the same symbols and production rules, their APTs can be directly compared. To make it easier to visually distinguish between plots of AETs and APTs, we plot AETs in a circular layout and APTs in a top-down layout.

Figure 4.1 demonstrates the composing of an aggregated parse tree visualization on the simple example of the linear grammar and the parse trees for expressions $x + y$ and $x + y + y$ from Chapter 2. The first and second image depict the parse trees for each of two expressions. The third image shows the aggregated parse tree, obtained by performing the above procedure on the two parse trees. The nodes and edges that appear in both parse trees are solid, whereas the nodes and edges that appear in only one of the two parse trees have their transparency halved.

To visualize the constriction of the search space, imposed by attribute rules, we prepare a PAG and its PCFG counterpart by stripping the PAG of attribute rules. We then

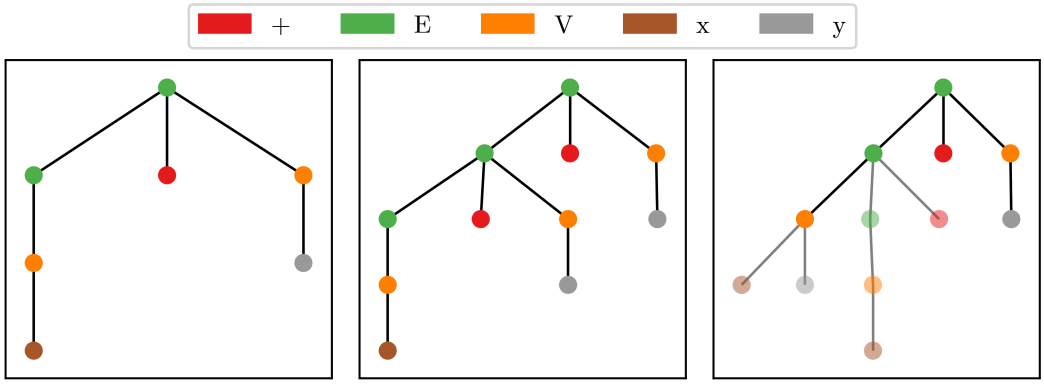


Figure 4.1: Demonstration of the composition of c) an aggregated parse tree from the individual parse trees of expressions a) $x + y$ and b) $x + y + y$, obtained using the linear grammar from Equation (2.1). Node colors correspond to individual nonterminal symbols. The transparency of nodes and edges corresponds to the normalized frequency of the respective derivation paths in collection of parse trees that form the aggregated parse tree.

generate N expressions using Algorithm 4.1 and Algorithm 1, respectively. Finally, we use the sampled expressions to compose an aggregated parse tree for each of the two grammars. Figure 4.2 demonstrates this comparison on the example grammar from Equation (4.1) – a polynomial grammar using attributes to generate only even powers up to the power of 10 – together with the corresponding AETs. The aggregated parse trees depicted in Figure 4.2 do not depict terminal symbols to improve readability. Comparing the two aggregated parse trees reveals the effect that the attribute rules have on the size of the search space. We obtained the aggregated parse trees by generating 1000 random expressions using each grammar. Note that the collection of expressions obtained this way contains many duplicates. In this case, 555 unique expressions were generated using the PCFG and only 33 using the PAG. Each of these expressions may be represented by a number of different parse trees, which are addressed individually. However, due to the existence of duplicates among the parse trees, the collections still contain fewer than 1000 parse trees. Nevertheless, the visualization can provide valuable insight into the structure and size of the search spaces of different PAGs.

4.4 Example: Dimensionally-Consistent Expressions

For the first example, we return to the now familiar problem of equipping arbitrary PCFGs with attribute rules that ensure dimensional consistency. We introduce only a single, inherited, attribute – the measurement unit in a vector form. Since we are working only with inherited attributes, dimensionally-consistent grammars require only pre-selection attribute rules. We begin with a simple example that demonstrates the key concepts – a grammar that generates dimensionally-consistent polynomials for the familiar problem of discovering $x = at^2$:

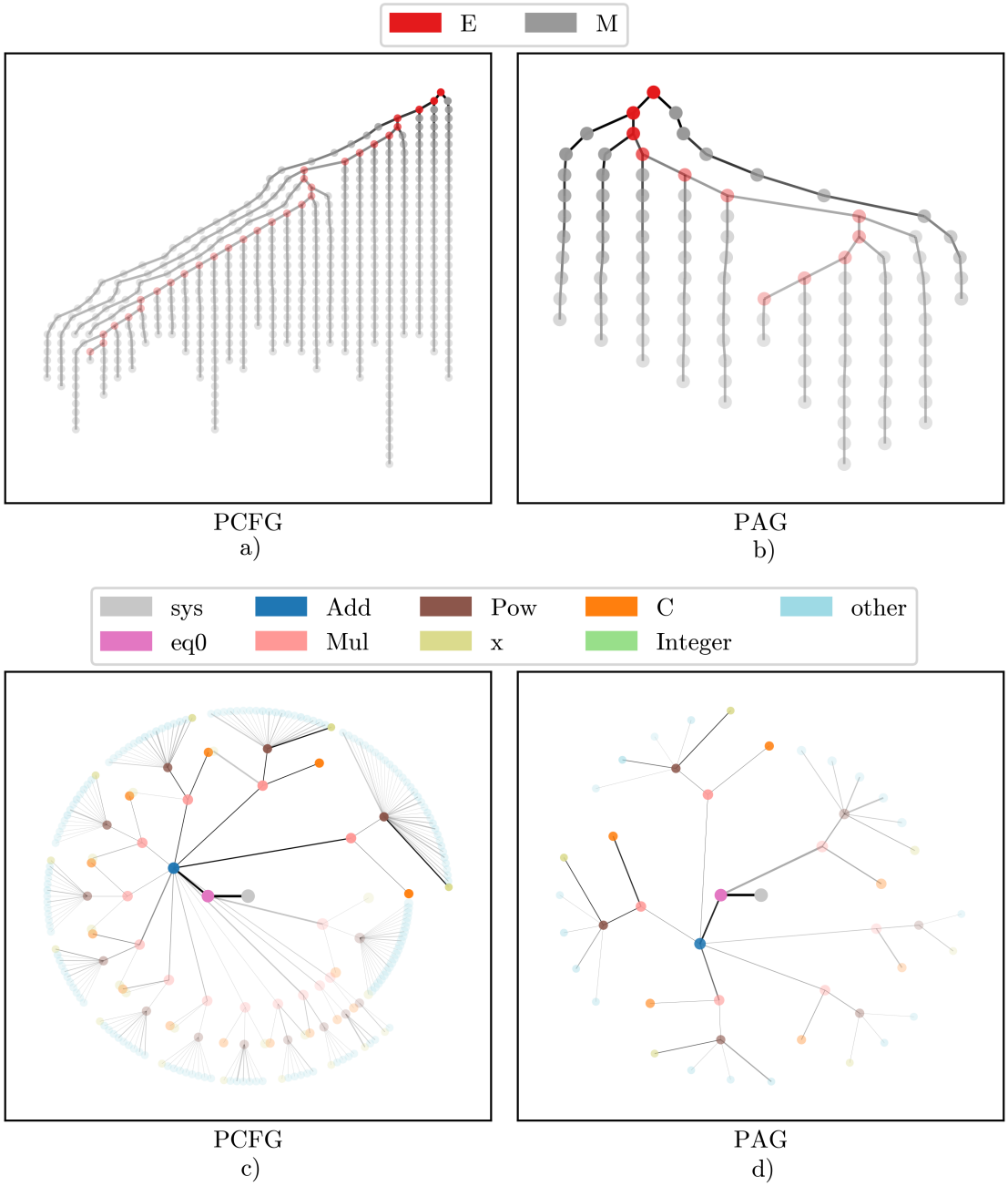


Figure 4.2: a) the aggregated parse tree and b) the aggregated expression tree of a **polynomial** PAG using attributes to constrain terms to even powers up to the power of 10 (Equation (4.1)), as well as c) the aggregated parse tree and d) the aggregated expression tree of the PCFG counterpart to the PAG. The aggregated trees were obtained by generating 1000 expressions with each grammar. Terminal symbols have been omitted from the APT to improve readability. The transparency of nodes and edges corresponds to the normalized frequency of the nodes and edges in the collections of parse trees or expression trees that form the APTs or AETs, respectively.

$$
\begin{array}{lcl}
P & \rightarrow & P + c * M \ [p_P] \ \{\text{"P2.u = P1.u; M1.u = P1.u"},,,\} \\
P & \rightarrow & c * M \ [1 - p_P] \ \{\text{"M1.u = P1.u"},,,\} \\
M & \rightarrow & M * V \ [p_M] \ \{\text{"V1.u = gen\_u(); M2.u = M1.u - V1.u"},,,\} \\
M & \rightarrow & V \ [1 - p_M] \ \{\text{"V1.u = M1.u"},,,\} \\
V & \rightarrow & a \ [0.5] \ \{,\text{"V1.u == array([1,-2])"},,\} \\
V & \rightarrow & t \ [0.5] \ \{,\text{"V1.u == array(0,1])"},,\}.
\end{array}
\quad , \tag{4.4}
$$

As in Section 1, the blue color highlights assignment-type pre-selection rules and the olive color the condition-type pre-selection rules. For this grammar, we define one global, the function *gen_ u*, which randomly selects one of the two two unit vectors $[0, 1]$ and $[1, -2]$. We set the starting symbol to P and its attribute u to the unit vector of the variable on the left-hand side of the equation we wish to discover: $P.u = \text{array}([1, 0])$.

The first production rule for M (multiplication) is particularly interesting. In Chapter 3, this production rule caused a lot of trouble during the transformation to a PCFG due to missing intermediate units, which required the introduction of auxiliary units. The direct sampling algorithm avoids this issue entirely. Since the algorithm does not rely on enumerating a finite selection of possible attribute values, the algorithm can handle attributes with arbitrary values. However, sampling such a grammar still fails often for certain problems. For example, any random derivation for at^2 will be rejected, if the production $M \rightarrow M * V$ is not chosen exactly once. Nonetheless, the failures should be less frequent than when transforming a PAG into a PCFG.

During the generation of a random expression with this grammar, any term with two variables (a derivation path in which $M \rightarrow M * V$ is chosen exactly once for each P) will successfully generate some permutation of $a * t * t$, regardless of which of the two units is generated by gen_u. Failures will occur only when terms with exactly one variable or more than two variables are generated. However, it must be noted that it is critical to assign the randomly chosen unit to the nonterminal V and the unit computed as $M1.u - V1.u$ to the nonterminal M. This order guarantees that the unit of V is an element of *units*, whereas the computed unit can be expanded into other units later in the generation process. The opposite assignment would guarantee a generation failure for this problem, unless auxiliary units were added. This generalizes as a rule of thumb for designing dimensionally-consistent PAGs. We can now consider a more complex dimensionally-consistent grammar – the

universal grammar for mathematical expressions:

$$
\begin{array}{lcl}
E & \rightarrow & E + F \ [p_{\text{sum}}] \ \{\text{"E2.u = E1.u; F1.u = E1.u"},,,\} \\
E & \rightarrow & E - F \ [p_{\text{dif}}] \ \{\text{"E2.u = E1.u; F1.u = E1.u"},,,\} \\
E & \rightarrow & F \ [1 - p_{\text{sum}} - p_{\text{dif}}] \ \{\text{"F1.u = E1.u"},,,\} \\
F & \rightarrow & F * T \ [p_{\text{mul}}] \ \{\text{"T1.u = gen\_u(); F2.u = F1.u - T1.u"},,,\} \\
F & \rightarrow & F/T \ [p_/] \ \{\text{"T1.u = gen\_u(); F2.u = F1.u + T1.u"},,,\} \\
F & \rightarrow & T \ [1 - p_* - p_/] \ \{\text{"T1.u = F1.u"},,,\} \\
T & \rightarrow & V \ [p_V] \ \{\text{"V1.u = T1.u"},,,\} \\
T & \rightarrow & C \ [p_C] \ \{,\text{"all(T1.u == zero)"},,\} \\
T & \rightarrow & R \ [1 - p_V - p_C] \ \{\text{"R1.u = T1.u"},,,\} \\
R & \rightarrow & (E) \ [p_f E] \ \{\text{"E1.u = R1.u"},,,\} \\
R & \rightarrow & sqrt(E) \ [p_f s] \ \{\text{"E1.u = 2*R1.u"},,,\} \\
R & \rightarrow & f_1(E) \ [p_f 1] \ \{\text{"E1.u = R1.u"},\text{"all(R1.u == zero)"},,\} \\
 & \dots & \\
R & \rightarrow & f_{n_f}(E) \ [p_{n_f}] \ \{\text{"E1.u = R1.u"},\text{"all(R1.u == zero)"},,\} \\
V & \rightarrow & v_1 \ [p_0] \ \{,\text{"V1.u == units[0]"},,\} \\
 & \dots & \\
V & \rightarrow & v_m \ [p_m] \ \{,\text{"V1.u == units[m-1]"},,\}.
\end{array}
\quad , \qquad (4.5)
$$

Where m is the number of variables and $\sum_{i=1}^{m} p_i = 1$, and $f_1 \dots f_{n_f}$ is the chosen set of special functions, such as the exponential function and trigonometric functions. The globals required are

1. *units*: list of numpy arrays, representing measurement units,
2. *gen_u*: function that returns random unit vector from *units*,
3. *zero*: numpy array, composed of a number of zeros equal to the number of basic units.

The function *all*, used in many attribute rules, is a numpy function that returns true if all values in an array are true. This grammar has the same production rules as the universal PCFG, introduced in Equation (2.26), and uses similar attribute rules as the polynomial PAG in Equation (4.4). There are several new patterns worth looking at in more detail. Firstly, this grammar explicitly allows only dimensionless numerical constants in the second production rule for T. However, this is not a limitation of the formalism. When addressing a more specific equation discovery problem, domain knowledge might call for the use of a finite set of dimensioned constants. For example, in the case of accelerated motion, we might wish to include constants with units of position and velocity, which could improve the chances of discovering an equation incorporating initial position and velocity, such as $x = x_0 + v_0 t + 0.5at^2$. A production rule for the positional constant would follow the pattern:

$$
T \quad \rightarrow \quad Cv \ [p_{Cv}] \ \{,\text{"T.u == array([1,0])"},,\}. \qquad (4.6)
$$

Secondly, the new PAG formalism does not limit us to dimensionless arguments of special functions. The second production rule for R encodes the square root, which can take an expression with any unit as the argument. The assignment-type attribute rule $E.u = 2{*}R.u$

Table 4.1: Summary of experimental results, comparing the properties of two approaches to sampling dimensionally-consistent grammars. For each problem from the Feynman database, 10^5 expressions were generated using each approach. In the first row we report the percentage of successful samplings, averaged over the 100 problems. The second row gives the average time required to perform a single random generation, successful or not, on a desktop computer. In the third row, we report the number of problems from the Feynman database, for which no expressions were generated successfully.

grammar	**PAG**	**PCFG**
mean success rate	17.8%	17.2%
mean time (ms)	3.0	0.22
# of unsuccessful samplings	12	18

ensures that the output of the square root will have a unit that is half of the input's unit. We could write similar rules to explicitly include higher-order roots or power functions. On the other hand, in the grammar above, we keep the arguments of other special functions dimensionless through the condition-type attribute rule $all(E.u == zero)$.

We compare the APTs and AETs of the polynomial and the universal PAG with their PCFG counterparts for the problem of discovering $x = at^2$ in Figures 4.3 and 4.4. The plots reveal that dimensionally-consistent PAGs explore a significantly simpler search space than PCFGs. This difference is particularly prominent for dimensionally consistent grammars, since the attribute rules for measurement units predominantly affect the structure of the derivation and allow for a much smaller number of valid parse trees than the unrestricted PCFGs.

4.4.1 Comparison to dimensionally-consistent PCFGs

We have demonstrated how we can use the new PAG formalism, coupled with the novel direct sampling method for PAGs, to implement dimensionally consistent PAGs and sample them directly. In this way, we can generate dimensionally-consistent expressions directly, without enumerating attribute values and transforming the PAG into a PCFG. However, the sampling process still fails often for many problems. We perform a computational experiment to investigate the differences between the two approaches to generating dimensionally-consistent expressions:

1. directly sampling a universal mathematical PAG using Algorithm 4.1,
2. introducing auxiliary units using Algorithm 3.2, transforming the universal mathematical PAG into a PCFG using Algorithm 3.1 and sampling the PCFG using Algorithm 1.

For each problem from the Feynman database, we perform 10^5 random generations of dimensionally-consistent expressions using each of the two approaches. We study the probability of successfully generating a random expression, called success rate, across the 100 Feynman equations, as well as the time required to perform a single generation, successful or not. We summarize the results in Table 4.1. The mean success rate is very similar for both approaches. Since the main advantage of direct sampling over the transformation to a PCFG is the absence of issues with missing units, these results confirm that the heuristic procedure for introducing auxiliary units in Algorithm 3.2 addresses the issues well. In terms of computation time, it takes around ten times as long to directly sample the PAG than to sample the transformed PCFG. The difference is due to the additional computational effort required to execute and evaluate the attribute rules of every production rule

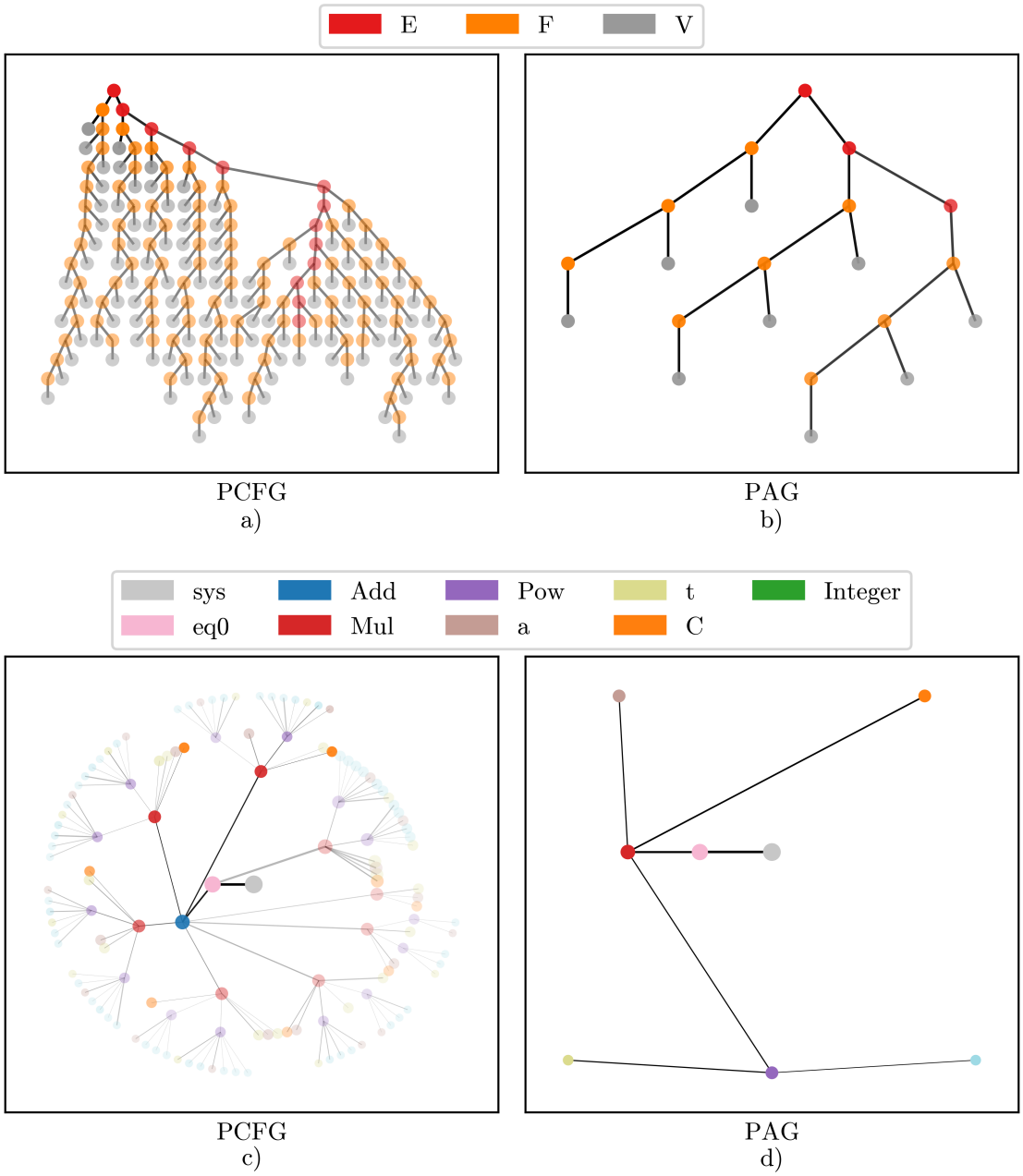


Figure 4.3: a) the aggregated parse tree and b) the aggregated expression tree of a **polynomial** PAG for the problem of discovering $x = at^2$, as well as c) the aggregated parse tree and d) the aggregated expression tree of the PCFG counterpart to the PAG. The aggregated trees were obtained by generating 1000 expressions with each grammar. Terminal symbols have been omitted from the APT to improve readability. The transparency of nodes and edges corresponds to the normalized frequency of the nodes and edges in the collections of parse trees or expression trees that form the APTs or AETs, respectively.

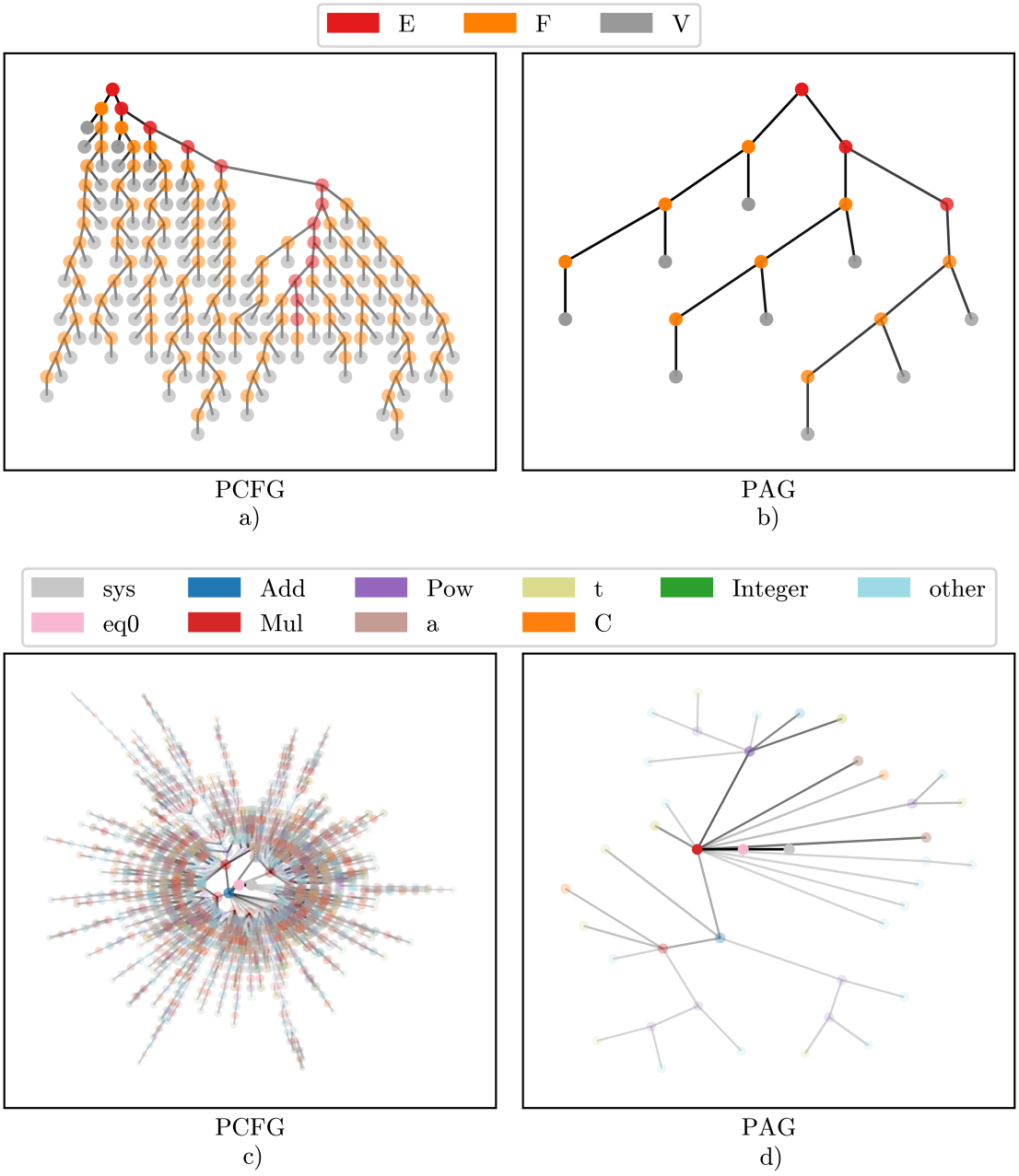


Figure 4.4: a) the aggregated parse tree and b) the aggregated expression tree of a **universal** PAG for the problem of discovering $x = at^2$, as well as c) the aggregated parse tree and d) the aggregated expression tree of the PCFG counterpart to the PAG. The aggregated trees were obtained by generating 1000 expressions with each grammar. Terminal symbols have been omitted from the APT to improve readability. The transparency of nodes and edges corresponds to the normalized frequency of the nodes and edges in the collections of parse trees or expression trees that form the APTs or AETs, respectively.

at every recursive step of the generation. On the other hand, for $N = 10^5$ we were unable to successfully generate any expressions for 12 problems by directly sampling the PAG, compared to the 18 unsuccessful samplings for the PCFG. The most likely explanation is that Algorithm 3.2 does not work for the six problems where the PAG succeeds and the PCFG does not. However, the difference could also be statistically insignificant, since the number of expressions generated with the PAG was very low for these problems.

Overall, the two approaches perform similarly on the Feynman dataset, although the direct PAG sampling method is significantly slower. However, since the time required to generate random expressions is generally very small compared to the time required to estimate their parameters, the difference in speed is not concerning. The procedures introduced in Algorithm 3.1 and Algorithm 3.2 are specific to PAG for dimensionally-consistent equations. Our main purpose in developing a novel direct PAG sampling method is to enable the use of PAG for expressing various types of background knowledge, including, but not limited to, dimensional consistency. In this section, we have demonstrated how the new PAG formalism, coupled with Algorithm 4.1, addresses dimensional consistency.

4.5 Example: Dynamical Systems

Dynamical systems theory is a broad and important area within mathematics, concerned with the study of systems that evolve over time according to a set of rules, typically represented by differential equations. The primary aim is to understand and describe the behavior of these systems based on their initial states and governing equations. The mathematical concept of a dynamical system provides an abstract framework to model and analyze physical, biological, or even economical systems undergoing change. Common examples include the study of planetary motion in celestial mechanics, population dynamics in ecology, or stock market fluctuations in economics. At the core of dynamical systems theory is the representation of a studied system as a system of ordinary or partial differential equations (ODEs or PDEs). These equations govern the temporal evolution of system variables, which can depend on factors such as time, space, or other system variables. Solutions to these equations offer a time-series output of the system's behavior and can depict fixed points, periodic oscillations, or chaotic trajectories. Equation discovery in the context of dynamical systems involves identifying these governing equations from observational data, which can lead to a deeper understanding of the system's inherent dynamics, potentially enabling improved predictions of system behavior and even scientific discoveries.

Dynamical systems require a minor extension of the equation discovery framework we have been using so far, since they are represented not by a single equation, but rather a system of equations. The simplest solution is to use a grammar for mathematical expressions to randomly generate one expression for each of the equations in the system. However, this approach is naive, since the structures of the equations forming a system of ODEs are rarely independent from each other. Rather, the system tends to have an overall structure and often features terms that appear in several equations. To address this, we can design grammars that generate the entire system of equations as a tuple of expressions. We can easily implement this by using a production for the starting symbol, following the pattern:

$$S \to E, \ldots, E[1.0],$$

where S is the starting symbol and E is a symbol that the grammar expands into a mathematical expression. The production features a number of symbols E, equal to the number of equations in the system of ODEs.

4.5.1 Coupling terms

In systems of ordinary differential equations (ODEs) describing dynamical systems, coupling terms play a crucial role. These terms essentially represent interactions between different components or variables of the system. Consider the following examples of dynamical systems featuring coupling terms.

1. Lotka-Volterra equations (ecology). In the predator-prey model described by the Lotka-Volterra equations, the term involving the product of the two populations xy is a nonlinear coupling term. It represents the rate at which predators y eat prey x.

$$\begin{aligned} \dot{x} &= ax - bxy, \\ \dot{y} &= -cy + dxy. \end{aligned} \tag{4.7}$$

2. Brusselator model (chemical kinetics). The Brusselator is a theoretical model for a type of autocatalytic reaction. The equations contain the same nonlinear coupling term xy^2 in both equations:

$$\begin{aligned} \dot{x} &= A + x^2y - Bx - x, \\ \dot{y} &= Bx - x^2y. \end{aligned} \tag{4.8}$$

Coupling terms often, but not always, appear in the same form in several equations in a system of ODEs. This represents a problem when using PCFGs as generators of expressions. Although we can write a PCFG that generates an entire system of ODEs and include productions that specifically derive coupling terms, we have no way of ensuring that a term, generated for one equation, is reused in the other equation.

We can, however, impose this restriction using attribute grammars by introducing two new attributes *frozen*(boolean) and *subtree*(string). These two attributes are attached to each nonterminal symbol in our PAG formulation and have a special function in the direct sampling algorithm (Algorithm 4.1). When beginning the derivation of a given nonterminal symbol, the procedure first checks whether it is frozen. If yes, the method simply returns the stored *subtree* of the nonterminal symbol, instead of performing the random generation. By default, no nonterminal symbol is frozen. We can freeze it by setting *frozen* to true in an appropriate post-selection assignment-type attribute rule. In the case of dynamical systems, we can randomly generate the structure of a coupling term, freeze the corresponding nonterminal symbol, and simply reuse the generated structure later, when we derive the expressions of other equations in the system. To demonstrate this use case, we design a PAG for dynamical systems with two state variables that generates systems of ODEs with the following structure:

$$\begin{aligned} \dot{x} &= P_1(x) + M(x, y), \\ \dot{y} &= P_2(y) + M(x, y), \end{aligned} \tag{4.9}$$

where P_1 and P_2 are polynomials and M is a monomial. Equation (4.10) presents a PAG, implementing the two main restrictions: 1) each equation should be composed of a polynomial of the corresponding variable and a monomial of both variables, 2) the monomial

should take the same form in both equations.

$$
\begin{array}{rcl}
S & \to & P, P\ [1.0]\ \{\text{"P1.v = 'x'; P2.v = 'y'"},,,\} \\
P & \to & P + c * M\ [p_P]\ \{\text{"P2.v = P1.v; M1.v = P1.v"},,,\} \\
P & \to & c * M + c * C\ [1 - p_P]\ \{\text{"M1.v = P1.v"},,,\} \\
M & \to & M * V\ [p_M]\ \{\text{"M2.v = M1.m; V1.v = M1.v"},,,\} \\
M & \to & V\ [1 - p_M]\ \{\text{"V1.v = M1.v"},,,\} \\
V & \to & x\ [0.5]\ \{,\text{"V1.v == 'x'"},,\} \\
V & \to & y\ [0.5]\ \{,\text{"V1.v == 'y'"},,\} \\
C & \to & Mc\ [1.0]\ \{,,\text{"C.frozen = True"},\} \\
Mc & \to & Mc * Vc\ [p_C]\ \{,,,\} \\
Mc & \to & Vc * Vc\ [1 - p_C]\ \{,,,\} \\
Vc & \to & x\ [p_x]\ \{,,,\} \\
Vc & \to & y\ [1 - p_x]\ \{,,,\}.
\end{array}
\qquad (4.10)
$$

The first production rule generates a tuple of expressions. The pre-selection assignment-type statements introduce an attribute v, which we will use to track whether we are deriving $\dot{x}$ or $\dot{y}$. The next six production rules (for nonterminals P, M and V) are essentially a polynomial grammar with two modifications. The first difference is that we use pre-selection assignment-type rules to pass the attribute v down the parse tree to the nonterminal V. In the production rules for V, we use the information in the attribute v in the pre-selection condition-type rules to allow the selection of only the variable, relevant to the equation we are deriving. The second addition to the polynomial grammar is the term $c * C$ in the second production rule for P, which will derive the coupling term. The placement of $c * C$ ensures that each equation has only one coupling term.

The production rule for C is simple – it merely replaces the nonterminal C with the nonterminal Mc. However, the post-selection assignment-type rule is crucial, since it performs the function of freezing the nonterminal C, after the expression for the nonterminal has been derived. The rest of the grammar is dedicated to generating the structure of the coupling term as a monomial of at least second order, without using any further attribute rules. Note that although the introduction of the intermediary nonterminal C may seem redundant, it is necessary for freezing to work correctly. If we placed $Mc.frozen = True$ into the post-selection assignment-type rules of Mc, we would run into an issue – since the production rule is recursive, there would be multiple instances of Mc, all trying to define the subtree of the global Mc, resulting in conflicts. Freezing nonterminals therefore typically requires intermediary nonterminal symbols and corresponding non-recursive production rules. Below are some examples of dynamical systems, generated by this grammar:

1) $\dot{x} = c_0x^5 + c_1x^3 + c_2xy + c_3x$ $\dot{y} = c_4xy + c_5y$	2) $\dot{x} = c_0x^2y^3 + c_1x$ $\dot{y} = c_2x^2y^3 + c_3y$
3) $\dot{x} = c_0x^2 + c_1x + c_2y^2$ $\dot{y} = c_3y^2 + c_4y$	4) $\dot{x} = c_0x^4 + c_1x + c_2xy^2$ $\dot{y} = c_3xy^2$

Note that the third example features a "coupling" term y^2. In this example, our grammar derived a coupling term of second order, but chose y for both factors in the monomial. We could improve the PAG by introducing further attributes and attribute rules in the

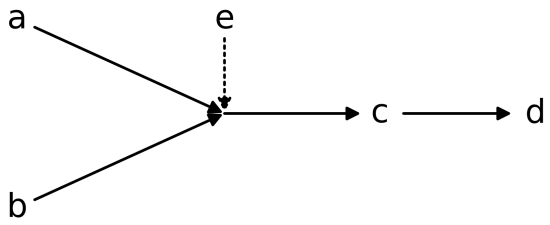


Figure 4.5: Example chemical reaction network involving the concentrations of four reactants (a, b, c, d) and an enzyme (e), connected by two chemical reactions ($A + B \to C$, $C \to D$).

last four production rules that ensure both variables appear in the coupling term. This grammar has a further weakness in that it cannot generate coupling terms like $x - y$ or $\sin(x - y)$. We could fix this by expanding the production rules for the coupling term to allow for more types of expressions.

4.5.2 Chemical kinetics

Chemical kinetics is a specialized subfield of physical chemistry, primarily concerned with the detailed study of reaction rates and the factors influencing them. It focuses on the temporal behaviors of chemical reactions, which entails the quantification of reaction rates and the examination of how different variables, such as temperature, pressure, and the concentration of participating substances, affect these rates. A key aspect of chemical kinetics involves the mechanistic dissection of reactions, where the stepwise sequence of elementary reactions leading to the overall reaction is analyzed. This examination aids in the derivation of the rate law for the reaction, an equation that directly links the rate of reaction to the concentrations of the reactants. The theoretical underpinnings of chemical kinetics rely heavily on a system of first-order ordinary differential equations (ODEs). These equations manifest as rate laws, describing the relationship between the change in concentration of reactants and products over time and the rate constants. The latter are empirically determined constants that indicate the speed of a specific reaction under certain conditions. The discovery and subsequent understanding of these equations play a pivotal role in chemical kinetics. Figure 4.5 is a schematic, depicting an example chemical reaction network involving the concentrations of four reactants (a, b, c, d) and an enzyme (e), connected by two chemical reactions ($A + B \to C$, $C \to D$). The reaction network is described by the following system of differential equations:

$$\begin{aligned} \dot{a} &= C_1 \cdot e \cdot a^{\gamma_{1a}} \cdot b^{\gamma_{1b}} \\ \dot{b} &= C_2 \cdot e \cdot a^{\gamma_{1a}} \cdot b^{\gamma_{1b}} \\ \dot{c} &= C_3 \cdot e \cdot a^{\gamma_{1a}} \cdot b^{\gamma_{1b}} + C_4 \cdot c^{\gamma_{2c}} \\ \dot{d} &= C_5 \cdot c^{\gamma_{2c}}. \end{aligned} \tag{4.11}$$

Here, $\gamma_{1a}, \gamma_{1b}, \gamma_{2c}$, as well as $k_1, \ldots, k_5$ are numerical constants. Taking into account the reaction directions, we know that $C_1, C_2.C_4 < 0$ and $C_3, C_5 > 0$. We can now identify the following domain knowledge that can be used in equation discovery.

1. The reaction network is described by system of differential equations, one for each variable.

2. Each equation's right-hand side is a linear combination of terms.
3. Each term has the form $\prod_{j=0}^{k} v_j^{\gamma_{iv_j}}$, where v_j refers to the k-th state variable.
4. If a term contains variable v_j, the equation for $\dot{v_j}$ must contain this term.
5. Consequently, each term appears in at least k equations.
6. Each instance of a term in the system has its own numerical constant C, but shares the exponents γ_{iv_j} with the other instances of the same term.

We can encode this domain knowledge using probabilistic attribute grammars. In a general equation discovery application in chemical kinetics, we assume we know the number of reactants and enzymes involved. Furthermore, in this PAG, we limit the maximum number of possible chemical reactions. To demonstrate the grammar, we decide on the same parameters as in Figure 4.5: four reactants and at most two chemical reactions. The resulting PAG is relatively complicated, so we examine it step by step. The grammar begins with the production rule for the starting symbol S, which generates a system of four equations. We represent it as a tuple of four expressions, separated by commas:

$$S \quad \rightarrow \quad E1, E2, E3, E4\ [1.0]\ \{\text{"S.ET = [[],[],[],[]]; S.TV = [[],[]]"}, , , \text{"verify(S.ET, S.TV)"}\}.$$

To express the constraints, imposed by the domain knowledge in fourth item above, we initiate two attributes to the global S in the pre-selection assignment-type attribute rule. The first, $S.ET$, will contain an array for each equation in the system. Each of the arrays will be composed of indices, identifying the terms that appear in that equation. The second attribute, $S.TV$ will contain an array for each of the generated terms. Each of the arrays will be composed of indices, identifying the variables that appear in that term. Finally, in the post-selection condition-type attribute rule, we call *verify* – a global function that returns true only if for each reactant in each generated term, the term appears in the expression for the time derivative of the reactant concentration. Since this attribute rule is placed in the post-selection step of the first production, its evaluation will be the very last step in the generation of a random system of equations. At that point, all expressions and terms will have been generated and the information about the structure of the generated system will be summarized in $S.ET$ and $S.TV$ and subsequently used by *verify*. The next couple of production rules are fairly simple:

$$\begin{aligned}
E1 \quad &\rightarrow \quad E\ [1.0]\ \{\text{"E1.eqi = 0"}, , , \} \\
&\dots \\
E4 \quad &\rightarrow \quad E\ [1.0]\ \{\text{"E1.eqi = 3"}, , , \} \\
E \quad &\rightarrow \quad E + T\ [p_E]\ \{\text{"E2.eqi=E1.eqi; T1.eqi = E1.eqi"}, , , \} \\
E \quad &\rightarrow \quad T\ [1 - p_E]\ \{\text{"T1.eqi=E1.eqi"}, , , \}.
\end{aligned}$$

Here, the attribute *eqi* represents the integer index of the equation. The rules for $E1, E2, E3$ and $E4$ could have been included in the production rule for S. We split it into separate production rules to improve the readability of the grammar and to enable a more interesting visualization of the grammars aggregated parse tree later. The pre-selection assignment-type attribute rules pass the information on which equation from the system of ODEs we are deriving down the parse tree, using the attribute *eqi*. Meanwhile, the two production rules for E determine the number of terms in an expression, parameterized by the probability of recursion p_E, as well as pass the attribute *eqi* further down the parse tree. Next,

we introduce three pairs of production rules that manage the generation of terms. The first pair:

$$\begin{aligned} T &\rightarrow T1\ [0.5]\ \{,,\text{"S.ET[T1.eqi] += [0]"},\} \\ T &\rightarrow T2\ [0.5]\ \{,,\text{"S.ET[T1.eqi] += [1]"},\} \end{aligned}$$

decides (randomly) which of the two possible terms will be added to the current expression. The post-selection assignment-type attribute rules append the index of the chosen term to the array in $S.ET$ at index eqi (which corresponds to the current equation). This way, we keep track of which term is included in each of the equations. It is important to execute this assignment in the post-selection attribute rule. Had we included it in the pre-selection attribute rules, it would be executed twice – once for $T1$ and once for $T2$, the pre-selection rules are executed and evaluated for all applicable production rules, before selecting the production rule to be included in the derivation. The final pair of production rules that manages the terms is

$$\begin{aligned} T1 &\rightarrow M\ [1.0]\ \{\text{"M1.i = 0"},,\text{"T1.frozen = True"},\} \\ T2 &\rightarrow M\ [1.0]\ \{\text{"M1.i = 1"},,\text{"T2.frozen = True"},\}. \end{aligned}$$

Note that we have started with one nonterminal (T), split off into two nonterminals ($T1$ and $T2$) and now merge back into one nonterminal (M). The grammar must be structured this way to enable us to freeze a term after it is derived for the first time. The attribute i tracks the integer index of the term. The pre-selection assignment-type attribute rules pass the information on which of the two terms we are deriving down the parse tree. The post-selection assignment-type is executed after the term has been derived and freezes the term. This means that henceforth, whenever the frozen term is generated during the derivation of this system of ODEs, the algorithm will simply reuse the subtree of the frozen term. This concludes the set of production rules, concerned with the management of terms. It is time to generate the structure of each term, using the familiar recursive production rules for generating monomials:

$$\begin{aligned} M &\rightarrow M * F\ [p_M]\ \{\text{"M2.i = M1.i; F1.i = M1.i"},,,\} \\ M &\rightarrow C * z * F\ [1 - p_Z]\ \{\text{"F1.i = M1.i"},,,\}, \\ M &\rightarrow C * F\ [1 - p_M - 1 - p_Z]\ \{\text{"F1.i = M1.i"},,,\}, \end{aligned}$$

which determine the number of factors in the term (and consequently the number of reactants in the reaction), as well as generate enzyme factors in the term. The distribution is parameterized by the probability of recursion p_M and the enzyme probability p_e. We again employ pre-selection assignment-type attribute rules to pass the index of the term we are deriving down the parse tree. If we had more than one possible enzyme to include, we could replace e above with a nonterminal that randomly chooses the specific enzyme for this factor. The next production rule could have been embedded into the production rule for M, but we separate it for better readability:

$$F \rightarrow pow(V, \gamma\ Ki\ Kv)\ [1.0]\ \{\text{"V1.i = F1.i; Ki1.i = F1.i; Kv1.V = V1"},,,\},$$

where pow, (,) and γ are terminal symbols. This production rule derives a factor in the term as one of the variables to the power of the corresponding numerical constant. We compose the symbol for the numerical constant using two nonterminals: Ki is related to the index of the term we are deriving and Kv to the variable that will be chosen with the nonterminal V. Besides passing the index of the term forward, we employ a new trick in

the pre-selection assignment-type rule: we pass a reference to the nonterminal $V1$ to the nonterminal $Kv1$ as the attribute V. Next, we generate the variable for this factor:

$$\begin{aligned} V &\rightarrow a\ [0.25]\ \{, \text{"0 not in S.TV[V1.i]"}, \text{"S.TV[V1.i] += [0]"}, \} \\ &\dots \\ V &\rightarrow d\ [0.25]\ \{, \text{"3 not in S.TV[V1.i]"}, \text{"S.TV[V1.i] += [3]"}, \}. \end{aligned}$$

These four production rules randomly choose one of the reactants as the variable. However, we make use of a pre-selection condition-type attribute rule to constrain the selection somewhat. We avoid duplicating variables in terms by checking whether the index of each variable is already included in the last array in $S.TV$, which corresponds to the term we are deriving. Then, the post-selection assignment-type attribute rule appends the index of the selected variable to the same array. Finally, we generate the numerical constant for the exponent of the variable, which we earlier decomposed into $\gamma Ki Kv$:

$$\begin{aligned} Ki &\rightarrow 1\ [0.5]\ \{, \text{"Ki1.i == 0"}, , \} \\ Ki &\rightarrow 2\ [0.5]\ \{, \text{"Ki1.i == 1"}, , \} \\ Kv &\rightarrow a\ [0.25]\ \{, \text{"'a' in Kv1.V.subtree"}, , \} \\ &\dots \\ Kv &\rightarrow d\ [0.25]\ \{, \text{"'d' in Kv1.V.subtree"}, , \}. \end{aligned}$$

Here, the two production rules for Ki use the index of the term we have been passing down the parse tree to effectively transform the index from an attribute value to a terminal symbol. There are four production rules required to generate Kv, one for each reactant. Once again, the selection is in fact not random, since the pre-selection condition-type attribute rule checks the subtree of $V1$ (accessed through the reference stored in its attribute V) and allows only the production rule for the variable we have generated in $V1$. Two example systems of ODEs, generated using this grammar with two maximum reactions and $p_E = 0.5, p_M = 0.5, p_Z = 0.1$, are shown below:

$$\begin{aligned} \dot{a} &= C_0 b^{\gamma_{1b}} d^{\gamma_{1d}} + C_1 c^{\gamma_{2c}} \\ \dot{b} &= C_2 b^{\gamma_{1b}} d^{\gamma_{1d}} \\ \dot{c} &= C_3 c^{\gamma_{2c}} \\ \dot{d} &= C_4 b^{\gamma_{1b}} d^{\gamma_{1d}} + C_5 c^{\gamma_{2c}} \end{aligned} \qquad \begin{aligned} \dot{a} &= C_0 a^{\gamma_{2a}} b^{\gamma_{2b}} \\ \dot{b} &= C_1 a^{\gamma_{2a}} b^{\gamma_{2b}} + C_2 b^{\gamma_{1b}} d^{\gamma_{1d}} z \\ \dot{c} &= C_3 a^{\gamma_{2a}} b^{\gamma_{2b}} + C_4 b^{\gamma_{1b}} d^{\gamma_{1d}} z \\ \dot{d} &= C_5 b^{\gamma_{1b}} d^{\gamma_{1d}} z. \end{aligned}$$

By increasing the maximum number of reactions to three, we generate systems such as the following:

$$\begin{aligned} \dot{a} &= C_0 a^{K1a} c^{K1c} \\ \dot{b} &= C_1 b^{K3b} \\ \dot{c} &= C_2 a^{K1a} c^{K1c} + C_3 b^{K3b} + C_4 d^{K2d} z \\ \dot{d} &= C_5 a^{K1a} c^{K1c} + C_6 b^{K3b} + C_7 d^{K2d} z \end{aligned} \qquad \begin{aligned} \dot{a} &= C_0 a^{K1a} b^{K1b} c^{K1c} z + C_1 b^{K2b} d^{K2d} \\ \dot{b} &= C_2 a^{K1a} b^{K1b} c^{K1c} z + C_3 b^{K2b} d^{K2d} + C_4 d^{K3d} \\ \dot{c} &= C_5 a^{K1a} b^{K1b} c^{K1c} z \\ \dot{d} &= C_6 b^{K2b} d^{K2d} + C_7 d^{K3d}. \end{aligned}$$

We can see that the generated systems of ODEs follow all the points of domain knowledge we have identified. In an actual equation discovery problem, this would result in a more constricted search space, requiring fewer evaluations of candidate systems, thereby improving the chance of success and improving the computational efficiency of equation discovery. Figure 4.6 compares the APTs and AETs of the presented PAG for chemical kinetics with its PCFG counterpart. In this case, the visual difference is not as obvious as

in the dimensional consistency example. This is because the main differences are in the selection of the generated terminals, which would be difficult to discern in the visualization. Nevertheless, the aggregated parse tree for the PAG is noticeably more ordered and structured than the aggregated parse tree for the PCFG, and the AET for the PAG is notably more constrained than the AET for the PCFG.

4.6 Example: Electronic Circuits

The field of electronics represents a vital aspect of modern engineering and technology. Research and development in electronics often involves the study, modeling, design and even identification of electric circuits. An electric circuit is an interconnection of electrical elements such as resistors, capacitors, amplifiers, transistors, etc. The ability to model these circuits accurately is crucial to designing and optimizing electronic devices for efficiency, reliability, and performance. Circuit modeling involves deriving mathematical representations of circuit behavior, often expressed as a system of differential equations. Circuit identification, on the other hand, typically involves the empirical determination of circuit's parameters, such as resistance, capacitance, or inductance values, typically based on measured responses to known inputs. Modern system identification approaches, such as equation discovery, however, allow us to go beyond the determination of parameters and enable the discovery of the structure and topology of a circuit.

Identifying electronic circuits represents a considerable challenge for equation discovery, in part due to the wide variety of relevant physical quantities and the relations between them, and more importantly, due to the variety and complexity of the topology that the structure of a circuit can exhibit. Due to these difficulties, electronic circuits are a prime candidate for demonstrating the use of PAGs to express complex types of domain knowledge.

4.6.1 RLC circuits

For our example, we will limit ourselves to so-called RLC circuits, which consist of voltage or current sources, resistors (R), inductors (L) and capacitors (C) in various configurations. Their properties render RLC circuits instrumental in a variety of applications, including tuning in radio devices, filtering signals and managing energy in power systems. Note that there is a number of relevant system identification problems we could address, involving RLC circuits. We choose to focus on one of them, with the chief purpose of demonstrating the use of PAGs in a complex manner. We model the RCL circuit as a dynamical system, described by a set of state variables, composed of the voltage across each capacitor and the current through each inductor. The current i_C through a capacitor is directly proportional to the time derivative of its voltage u_C:

$$i_C(t) = C\dot{u_C}, \tag{4.12}$$

where C is a numerical constant – the capacitance of the capacitor. Similarly, the voltage u_L across an inductor can be calculated as the time derivative of its current i_L:

$$u_L(t) = L\dot{i_L}, \tag{4.13}$$

where L is the inductivity of the inductor. Because these relations involve time derivatives, the dynamics of a RLC circuit are governed by a system of 1st order ODEs. Besides capacitors and inductors, RLC circuits contain any number of resistors, which do not introduce state variables and follow Ohm's law:

$$u_R = Ri_R, \tag{4.14}$$

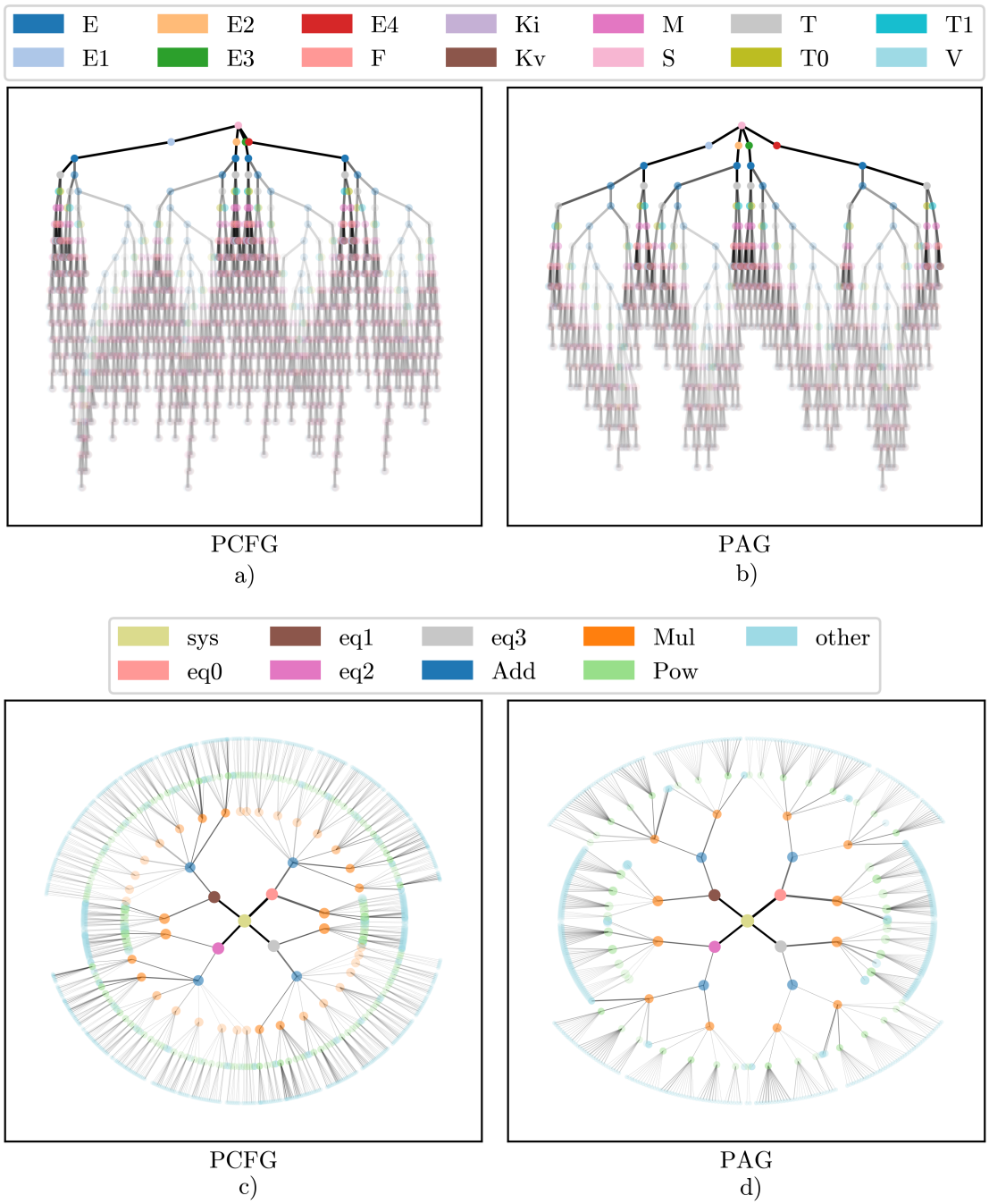


Figure 4.6: a) the aggregated parse tree and b) the aggregated expression tree of a PAG for generating systems of ODEs that follow the domain knowledge of chemical kinetics, as well as c) the aggregated parse tree and d) the aggregated expression tree of the PCFG counterpart to the PAG. The aggregated trees were obtained by generating 1000 expressions with each grammar. Terminal symbols have been omitted from the APT to improve readability. The transparency of nodes and edges corresponds to the normalized frequency of the nodes and edges in the collections of parse trees or expression trees that form the APTs or AETs, respectively.

where u_R is the voltage across the resistor, i_R is the current through the resistor and R is a numerical constant – the resistance of the resistor. The final element to include is a voltage source. A voltage source is an idealized approximation of a power source (such as a battery) that generates a constant voltage u_G, but an undefined current. It has a counterpart, called a current source, which generates a constant current, but has an undefined voltage. More realistic approximations are obtained by coupling a voltage or current source with a resistor. In our example, we limit ourselves to voltage sources. The main mathematical tool used to derive the system of ODEs for complex circuits are the two Kirchoff's laws:

- **Kirchoff's current law** states that the sum of all currents flowing in or out of a junction must be zero: $\sum_{k=1}^{n} i_k(t) = 0$,
- **Kirchoff's voltage law** states the the sum of all voltages in any closed loop in the circuit must be zero $\sum_{k=1}^{n} u_k(t) = 0$.

4.6.2 Derivation example

As an example, consider the circuit in Figure 4.6.2. We apply Kirchoff's voltage law on two different loops. The first loop starts at the voltage source and passes through the resistor and capacitor before connecting back to the voltage source. This gives us the equation:

$$u_G - Ri_R - u_c = 0, \tag{4.15}$$

where u_G is the voltage of the source, R is the resistance, i_R is the current through the resistor (which is the same as the current through the source) and u_c is the voltage on the capacitor. Note that the sign of each term is based on the direction we pass a component in. The details of how to choose the signs are beyond the scope of this work, but a general rule of thumb is that any convention will yield the correct results, as long as we apply it consistently. The second loop, which contains the capacitor and the inductor, gives the following equation:

$$u_C = L\dot{i}_L, \tag{4.16}$$

where L is the inductivity and i_L the current through the inductor. This equation readily provides one of the two differential equations we are deriving: $\dot{i}_L = \frac{1}{L}u_C$. Next, we apply Kirchoff's current law to any of the two junctions:

$$-i_R + C\dot{u}_C + i_L = 0. \tag{4.17}$$

Similar as before, we obtained the signs by choosing the directions of all the currents in the circuit, making sure the flow is consistent, then applying a negative sign to currents that flow out of the junction and a positive sign to currents that flow into the junction. We can now combine Equations (4.15) and (4.17) to eliminate i_R and obtain the differential equation for $\dot{u}_C$. The system of ODEs, describing the circuit, is:

$$\begin{aligned} \dot{u}_C &= \frac{1}{RC}(u_G - u_C) - \frac{1}{C}i_L \\ \dot{i}_L &= \frac{1}{L}u_C. \end{aligned} \tag{4.18}$$

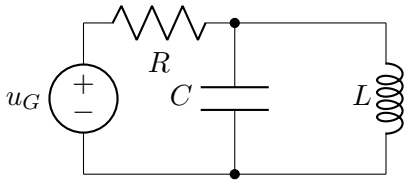


Figure 4.7: The diagram of an example electronic circuit, composed of a voltage source (u_G), a resistor (R), a capacitor (C) and an inductor (L).

4.6.3 PAGs for RLC circuits

Our goal in this section is to design a PAG that generates systems of ODEs, following the domain knowledge of RLC circuits. First, consider the different approaches to discovering such a system, ordered by the increasing level of complexity and amount of encoded domain knowledge.

1. **Sparse linear regression**. RLC circuits are described by linear systems of ODEs. This means that equation discovery methods based on L1 regularization, such as SINDy [3], are well suited for the problem. However, the results of this approach are not very interpretable, since they 1) do not reveal the topology of the circuit and 2) combine the original parameters of the circuit into generic numerical constants (i.e., instead of the symbolic expressions in Equation (4.18), we would obtain $\dot{u}_C = c_1 u_G + c_2 u_C + c_3 i_L$ and $\dot{i}_L = c_4 u_C$).

2. **Dimensionally-consistent expressions**. All the different quantities and constants involved in RLC circuits have unique, nontrivial measurement units. Dimensionally-consistent grammars are therefore a good candidate for this problem. Several aspects of the domain knowledge can readily be encoded by this approach. The linearity of the equations can be expressed in the structure of the grammar, instead of using generic numerical constants, a unique dimensioned constant can be added to the grammar for each generated component, while the physics is partially expressed by dimensional consistency.

3. **Grammars based on Kirchoff's laws**. The physics that electronic circuits obey is fully captured by the two Kirchoff's laws (at least in the approximation we are working with). Therefore, a grammar that ensures generated systems of ODEs follow Kirchoff's laws will generate only physically-correct candidates. In fact, for any given circuit (and a given set of state variables), such a grammar will be able to generate only a single system of ODEs – the correct one. System identification can then be performed by generating random circuits and their corresponding systems of ODEs, and evaluating their degree of fit on the observed data.

To test the limits of PAGs for expressing background knowledge, we design a system identification approach for RLC circuits, following the third approach. We assume that the measurements of all the state variables (or at least most state variables, since we can handle partially observed scenarios) are available. Consequently, we also know how many capacitors and inductors the circuit is composed of. The same goes for voltage sources. On the other hand, the circuit can contain any number of resistors.

The topology of the circuit is encoded in the attributes. Each component is represented by a global nonterminal, with the attribute *pins* – a list containing references to the global nonterminals it is directly connected to. We Resistors, capacitors, inductors and voltage sources have two pins. The first element of *pins* represents the positive (or input) pin

and the second element the negative (or output) pin. Besides the two-pin components, a circuit can contain any number of junctions. The *pins* attribute of a junction can have any number of elements. Junctions serve only as connections between other components and enable complicated circuit topologies. We generate a random circuit topology following Algorithm 4.2.

Algorithm 4.2: GENERATE_RANDOM_CIRCUIT(*twoPin*, p_{loop})
Generate random circuit topology from a list of components.

```
Data: List of 2-pin components twoPin, probability p_loop
Result: A random circuit graph with connections in twoPin and junctions
initialize junctions = [ ];
create two new junctions;
create the first loop by connecting: twoPin[0] → junctions[0] → twopin[1] →
  junctions[1] → twopin[0];
for i = 2; i < length(twoPin) do
    randomly decide whether to start a new loop or add to an existing one;
    if random() < p then
        choose two random junctions jun1, jun2 from junctions;
        create two new junctions new_jun1, new_jun2;
        add new loop by connecting: jun1 → new_jun1 → twoPin[i] → new_jun2
          → jun2;
    else
        choose random component comp from twoPin;
        next_comp = comp.pins[1];
        disconnect(comp, next_comp);
        connect(comp, twoPin[i]);
        connect(twoPin[i], next_comp);
    end
end
```

The circuits generated by Algorithm 4.2 can feature redundant complexity in the form of repeated components of the same type, connected in series. For example, consider a circuit with a single loop, containing a voltage source, two resistors and two inductors. Such a circuit can be simplified by removing one resistor and one inductor. We implement a simple procedure for simplifying circuits, which recursively traverses the circuit, detects repeated components in series and merges them. The complete procedure for generating a random candidate system of ODEs (and its associated circuit topology) has the following steps:

1. initialize a list of components *twoPin*, based on the fixed numbers of voltage sources, capacitors, inductors and resistors,
2. generate a random circuit by calling *generate_ circuit(twoPin,* p_{loop}*)* with the desired probability p_{loop},
3. simplify the generated circuit by removing repeated components, connected in series,
4. use the PAG to generate a system of ODEs, based on the circuit, encoded in the attributes.

Note that the PAG is fixed and can be reused for any circuit generated using a given combination of the numbers of voltage sources, capacitors, inductors and resistors. For simplicity, we present a PAG for circuits, composed of one capacitor, one inductor and one

resistor. It extends trivially to circuits with more components. The globals used by the grammar are:

- the global version of each nonterminal (required to freeze nonterminals),
- a "nonterminal" representing each capacitor (*Cap1, Cap2, ...*), inductor (*Ind1, Ind2, ...*), resistor (*Res1, Res2, ...*), voltage source (*Gen1, Gen2, ...*) and junction (*Jun1, Jun2, ...*),
- a list of references to each two-pin component, called *twoPin* and a list of references to each junction, called *junctions*.

The grammar begins with the production rule for the starting symbol, which derives the system of ODEs:

$$S \quad \rightarrow \quad duC1\ ,\ diL1\ [1.0]\ \{,,,\},$$

where $duC1$ and $diL1$ are nonterminals representing the time derivatives of the voltage across the capacitor and the current through the inductor, respectively. Next is a set of production rules that derive the expressions of all the unknowns: the time derivatives of each state variable, as well as the voltage and current of each resistor and the current through each voltage source. Note that only the time derivatives follow directly from S. The other unknowns are derived "on demand" as the grammar generates the corresponding nonterminals.

$$\begin{aligned}
duC1 &\rightarrow 1/C1 * (\ I\ )\ [1.0]\ \{\text{"I1.x = [Cap1]"},,\text{"duC1.frozen = True"},\}\\
diL1 &\rightarrow 1/L1 * (\ U\ )\ [1.0]\ \{\text{"U1.x = [Cap1]"},,\text{"diL1.frozen = True"},\}\\
IR1 &\rightarrow I\ [1.0]\ \{\text{"I1.x = [Res1]"},,\text{"IR1.frozen = True"},\}\\
UR1 &\rightarrow U\ [1.0]\ \{\text{"I1.x = [Res1]"},,\text{"UR1.frozen = True"},\}\\
IG1 &\rightarrow I\ [1.0]\ \{\text{"I1.x = [Gen1]"},,\text{"IG1.frozen = True"},\}.
\end{aligned}$$

Here, the right-hand side of each production rule contains a single nonterminal – either the general voltage U or current I, which will be derived in later productions. The pre-selection assignment-type rules of each production rule initialize an attribute x, which will be used to track the path of derivation through the circuit, and passes it down the parse tree. The post-selection assignment-type rule freezes the nonterminal once it has been derived. Note that for resistors, both voltage and current are unknown and must be derived, whereas for the other components, either the voltage or current is measured and the other is unknown. To expand the grammar for more components, we would add the appropriate production rules here. Next, we define the production rules that derive the voltage or current of any component required by the grammar. We find the voltage of the i-th component in a closed loop by rearranging Kirchoff's voltage law: $U_i = -\sum j \neq iU_j$. Thus, whenever we need to derive an unknown voltage, we begin a random walk in the circuit, which terminates successfully when it closes, or unsuccessfully when it runs into a

dead end. This functionality is performed by the following set of production rules:

$U \rightarrow PMu\ uC1\ U\ [1/N_c]$ {"U2.x = U1.x + [Cap1]; PM1.x=U2.x",
"Cap1 not in U1.x and Cap1 in U1.x[-1].pins",,}
$U \rightarrow PMu\ L1\ *\ (\ diL1\ )\ U\ [1/N_c]$ {"U2.x = U1.x + [Ind1]; PM1.x=U2.x",
"Ind1 not in U1.x and Ind1 in U1.x[-1].pins",,}
$U \rightarrow PMu\ R1\ *\ (\ IR1\ )\ U\ [1/N_c]$ {"U2.x = U1.x + [Res1]; PM1.x=U2.x",
"Res1 not in U1.x and Res1 in U1.x[-1].pins",,}
$U \rightarrow PMu\ uG1\ U\ [1/N_c]$ {"U2.x = U1.x + [Gen1]; PM1.x=U2.x",
"Gen1 not in U1.x and Gen1 in U1.x[-1].pins",,}
$U \rightarrow PMu\ U\ [1/N_c]$ {"U2.x = U1.x + [Jun1]; PM1.x=U2.x",
"Jun1 not in U1.x and Jun1 in U1.x[-1].pins",,}
...
$U \rightarrow PMu\ U\ [1/N_c]$ {"U2.x = U1.x + [JunN_J]; PM1.x=U2.x",
"JunN_J not in U1.x and JunN_J in U1.x[-1].pins",,}
$U \rightarrow$ [1.0] {,"U1.x[0] in U1.x[-1].pins and valid_closing(U1.x)",,},

where N_c is the total number of components and N_J is the number of junctions. The right-hand side of these production rules, except the last one, has a similar structure: "symbols U". The derivation of voltage for a given component entails recursively applying productions for the nonterminal U, until the final production rule above can be chosen, which closes the loop. The pre-selection assignment-type attribute rules of each production rule append the corresponding component to the path in x and pass it down the parse tree. The pre-selection condition-type rules check whether the corresponding component has been visited already this loop and whether it is connected to the previous component in the path – these rules are the crucial part of the grammar that takes into account the topology of the circuit. The grammar handles junctions naturally – the production rule of a junction adds nothing to the expression, it simply allows the algorithm to proceed to the next U, while appending the junction to the path. On the other hand, the final production rule in this set, which closes the loop, requires elaboration. Its right-hand side is empty, since the production rule serves only to end a loop. Its pre-selection condition allows it to be chosen only under strict circumstances – the current component must be connected to the component we started the loop with. Furthermore, to prevent trivial loops, the path must contain more than one two-pin component, and the current component must connect to the starting component using a different pin than we started the loop from. These conditions are implemented in the function valid_closing(path) for brevity. Finally, note that the right-hand side of some production rules in this set contains another nonterminal besides U, such as $diL1$ in the second production rule. Choosing such a production rule begins a new derivation of an unknown quantity, unless it has been frozen already. Lastly, each production rule in this set, except the last one, contains the nonterminal PMu ("plus-

minus"). This nonterminal is governed by the following production rules:

$$
\begin{aligned}
PMu \quad &\rightarrow \quad + \; [0.25] \; \{, \text{"PMu1.x[-1].pins.index(PMu1.x[-2]) == 0} \\
&\quad \text{"and PMu1.x[0].pins.index(PMu1.x[1]) == 0", , \}} \\
PMu \quad &\rightarrow \quad - \; [0.25] \; \{, \text{"PMu1.x[-1].pins.index(PMu1.x[-2]) == 1} \\
&\quad \text{"and PMu1.x[0].pins.index(PMu1.x[1]) == 0", , \}} \\
PMu \quad &\rightarrow \quad - \; [0.25] \; \{, \text{"PMu1.x[-1].pins.index(PMu1.x[-2]) == 0} \\
&\quad \text{"and PMu1.x[0].pins.index(PMu1.x[1]) == 1", , \}} \\
PMu \quad &\rightarrow \quad + \; [0.25] \; \{, \text{"PMu1.x[-1].pins.index(PMu1.x[-2]) == 1} \\
&\quad \text{"and PMu1.x[0].pins.index(PMu1.x[1]) == 1", , \}.}
\end{aligned}
$$

This set of production rules consistently implements a convention for setting the signs of terms in Kirchoff's voltage law. This convention sets the sign to plus if we connected to the current component through the first (negative) pin, and minus if we connected through the second (positive) pin. Furthermore, the sign is flipped if the loop started through the second (positive) pin of the starting component. The grammar we have shown so far can derive voltages by performing closed loops through the circuit, following the topology encoded in the *pin* attributes.

To derive the current of any component, the grammar performs a two-step procedure. First, it steps to a random neighbor of the component of interest and checks its current. If it is not a junction and its current is known, the job is done – the two currents are identical. If the current is not known, this procedure begins anew starting from that component. However, if the neighboring component is a junction, we apply a rearranged Kirchoff's current law for that junction: $I_i = \sum_{j \neq i} I_j$. In other words, we repeat this procedure for each other current flowing through the junction. This first part of this procedure is performed through the following set of production rules:

$$
\begin{aligned}
I \quad &\rightarrow \quad + \; C1 \; * \; (\; duC1 \;) \; [1/N_c] \; \{, \text{"Cap1 in I1.x[-1].pins", , \}} \\
I \quad &\rightarrow \quad + \; iL1 \; [1/N_c] \; \{, \text{"Ind1 in I1.x[-1].pins",,\}} \\
I \quad &\rightarrow \quad + \; (\; IR1 \;) \; [1/N_c] \; \{, \text{"Res1 in I1.x[-1].pins",,\}} \\
I \quad &\rightarrow \quad + \; (\; IG1 \;) \; [1/N_c] \; \{, \text{"Gen1 in I1.x[-1].pins",,\}} \\
I \quad &\rightarrow \quad IJ \; [1/N_c] \; \{\text{"IJ1.x = I1 + [Jun1]; IJ1.i=0", "Jun1 in I1.x[-1].pins",,\}} \\
&\ldots \\
I \quad &\rightarrow \quad IJ \; [1/N_c] \; \{\text{"IJ1.x = I1 + [Jun}N_J\text{]; IJ1.i=0", "Jun}N_J\text{ in I1.x[-1].pins",,\}.}
\end{aligned}
$$

These production rules have either no nonterminal (if the current is known) or one nonterminal (if the current is unknown) on the right-hand side. All but the junction production rules have only a single attribute rule that checks whether the corresponding component is connected to the previous component. The attribute x is expanded only when encountering a junction, in which case a new attribute is initialized as well – i will keep track of the wires as the next set of production rules iterates through the pins of the junction (note

that a junction can have any number of pins):

$IJ \rightarrow IJ\ IJi$ [0.5] {"IJ2.x=IJ1.x; IJ2.i=IJ1.i+1; IJi1.x=IJ1.x; IJi1.i=IJ1.i", "IJ1.i < len(IJ1.x[-1].pins)-1", , }

$IJ \rightarrow IJi$ [0.5] {"IJi1.x=IJ1.x; IJi1.i=IJ1.i", "IJ1.i >= len(IJ1.x[-1].pins)-1", , }

$IJi \rightarrow PMj\ Ij$ [0.5] {"Ij1.next=IJi1.x[-1].pins[IJi1.i]; Ij1.x=[IJi1.x[-1]]; PMj.x=IJi1.x+[IJj1.next]", "not IJi1.i == IJi1.x[-1].pins.index(IJi1.x[-2])", , }

$IJi \rightarrow$ [0.5] {, "IJi1.i == IJi1.x[-1].pins.index(IJi1.x[-2])", , }.

This set of production rules may seem complicated, but performs a rather simple function. The production rules for IJ generate one IJi for each pin in the junction, using the attributes x and i to keep track of the pins. The first production rule for IJi is chosen for all the pins except the one we accessed the junction with and generates the sign of the current and a new nonterminal Ij that derives the current flowing into (or out of) the pin. The assignment-type attribute rule stores the information on which pins current Ij should derive into the attribute *next*. The second production rule for IJi has an empty right-hand side and is used only for the pin we used to access the junction. In summary, this set of production rules expresses the reformulated Kirchoff's current law. The nonterminal PMj derives the sign of term, following a similar logic as PMu.

The final set of production rules needed for the grammar are those with Ij on the left-hand side. These are a copy of the production rules for I, with one difference. Instead of choosing any of the components connected to the starting component (in this case always a junction), they must choose the component specified in the attribute *next*. In other words, the condition rule "Cap1 in I1.x[-1].pins" is replaced by Cap1 == Ij1.next", and similarly for all other components.

To summarize, to randomly generate a physically-consistent system of ODEs, we first generate a random electronic circuit topology, encoded in the attributes of global nonterminals that correspond to circuit components. The PAG derives the differential equation for each state variable by recursively applying both Kirchoff's laws on unknown quantities. The generation process completes when all unknown quantities have been derived. We present two examples of generated systems of ODEs and the corresponding circuits below.

4.6.4 Discussion

The presented PAG for electronic circuits does not work perfectly. On the one hand, the PAG generates only a single expression for a given circuit – the correct one. However, for many generated topologies, the grammar is incapable of deriving any expressions. There are two ways Algorithm 4.1 using the PAG for electronic circuits can fail.

The first is encountering a dead-end during the derivation. This occurs because we apply Kirchoff's voltage law by performing a random walk through the circuit, hoping to eventually close the loop. Since we disallow returning to nodes that are already part of the loop, it is often possible to take a random path that cannot be closed under these restrictions. For more complicated topologies, the probability of successfully finishing a closed loop can be very small.

The second fail-state entails endless recursion, or more practically, encountering the recursion limit of the implementation. This occurs because the grammar effectively begins a new derivation of voltage or current whenever it encounters an unknown, even if it is

$$\dot{u}_{C1} = \frac{1}{C_1}(i_{L2} - i_{L1}),$$
$$\dot{u}_{C2} = \frac{1}{C_2} i_{L1},$$
$$\dot{i}_{L1} = \frac{R_1}{L_1}(i_{L2} - i_{L1}) + \frac{1}{L_1}(u_G - u_{C1} - u_{C2}),$$
$$\dot{i}_{L2} = \frac{R_1}{L_2}(i_{L1} - i_{L2}) + \frac{1}{L_2} u_G.$$

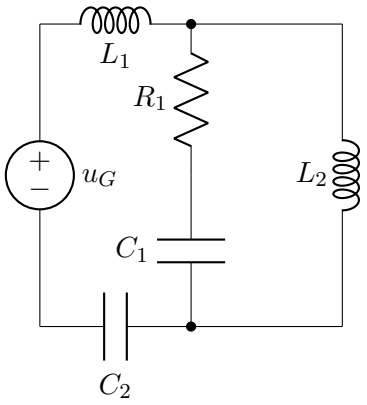


Figure 4.8: Example of a system of ODEs and the corresponding electronic circuit, generated using generate_circuit and the presented PAG for electronic circuits. The numbers of components were set to 2 capacitors, 2 inductors, 2 resistors and 1 voltage source. Note that some of the generated components have been removed during circuit simplification.

$$\dot{u}_{C1} = \frac{1}{C_1} i_{L2},$$
$$\dot{u}_{L1} = \frac{R_1}{L_1}(i_{L1} - i_{L2} - i_{L3}) - \frac{1}{L_1} u_{G2},$$
$$\dot{i}_{L2} = -\frac{R_1}{L_2}(i_{L1} - i_{L2} - i_{L3}) + \frac{1}{L_2}(u_{C1} + u_{G2})$$
$$\dot{i}_{L3} = \frac{R_1}{L_2}(i_{L1} - i_{L2} - i_{L3}) - \frac{1}{L_3}(u_{G1} + u_{G2}).$$

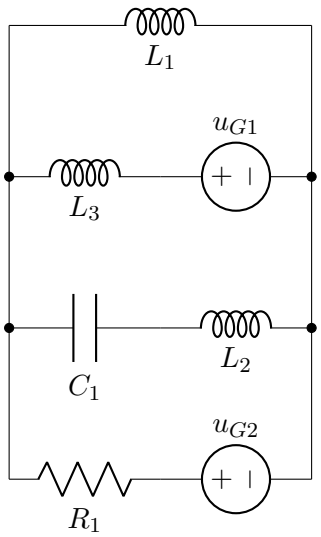


Figure 4.9: Example of a system of ODEs and the corresponding electronic circuit, generated using generate_circuit and the presented PAG for electronic circuits. The numbers of components were set to 2 capacitors, 3 inductors, 2 resistors and 2 voltage sources. Note that some of the generated components have been removed during circuit simplification.

Table 4.2: Results of the experiment investigating the sampling performance of the PAG for RLC circuits. The values represent the approximated probabilities that given a randomly generated RCL circuit, the approach successfully derives the correct system of ODEs in 100 tries (first row), the approach fails by recurring endlessly (second row) and the approach fails by reaching a dead end in the derivation (third row). The probabilities were approximated by randomly generating 100 RLC circuits for each configuration of the number of each component. Four different configurations were used for each total number of components and their results averaged.

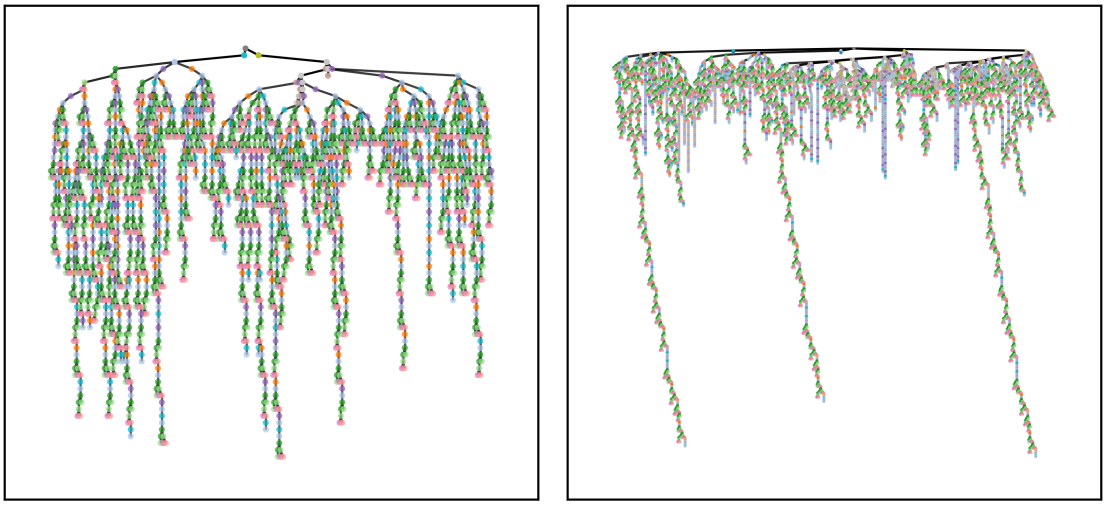
# of components	**3**	**4**	**5**	**6**	**7**	**8**	**9**
successful generation	61%	55%	36%	19%	10%	9%	8%
endless recursion	13%	7%	3%	1%	2%	3%	1%
reached dead end	26%	38%	61%	80%	88%	88%	91%

already in the process of deriving that very unknown. For example, this often occurs when capacitors are wired in parallel without any other components in-between. The derivation of the first capacitors current will require the current through the neighboring junction, which requires the current through the second capacitor. Following the same process, the current through the second capacitor will require the current though the first capacitor. This phenomenon is a fundamental weakness of the presented approach, which relies only on directly deriving each unknown with Kirchoff's laws.

To investigate the usefulness of the presented PAG, we perform an experiment, reported in Table 4.2. We generate 100 random circuits for different configurations of the number of each component. For each of the generated circuits, we give the algorithm 100 attempts to derive its system of ODEs. If any of the 100 attempts is successful, we consider this a **successful generation** – the grammar is capable of deriving the system of ODEs for that circuit. If none of the attempts are successful, we check whether the algorithm failed more often due to a dead end due to endless recursion. In other words, we classify each generated circuit into a success, a failure due to endless recursion or a failure due to a dead end, based on the results of 100 repeated attempts. We perform the experiment on 28 different configurations of the number of each component, four for each number of total components. For instance, for circuits with three components, we test the configurations $(n_C, n_L, n_R, n_G) \in \{(1,1,1,0), (1,1,0,1), (1,0,1,1), (0,1,1,1)\}$, where n_C, n_L, n_R and n_G are the number of capacitors, inductors, resistors and voltage sources, respectively. In Table 4.2, we report the proportion of each category (success, two types of failures) among the 100 randomly generated circuits, averaged across the four configurations with the same number of total components.

We can see that the proportion of circuits the PAG can successfully derive falls with the increasing number of components. This is expected, because more components means more unknowns, which increases the complexity of the derivation and the probability of failure. The probability of success ranges between 61% for the simplest circuits and 8% for the most complex ones. Unfortunately, this result is not promising, since these probabilities are quite low.

Surprisingly however, endless recursion accounts for only a very small proportion of the failures. Only between 1% and 13% of derivations terminate due to endless recursion. Furthermore, this number falls with the increasing number of components. This observation is promising, since issues with endless recursion are an inherent downside of the presented approach. In contrast, the dead end failures occur due to the simplistic and naive way we apply Kirchoff's voltage law. This type of error can be alleviated or even eliminated completely by improving the consideration of closed loops in the grammar. For instance,



4 two-pin components
a)

8 two-pin components
b)

Figure 4.10: Aggregated parse trees of PAGs for generating systems of ODEs that describe electronic circuits: a) the PAG for circuits with four two-pin components, b) the PAG for circuits with eight two-pin components. The aggregated parse trees were obtained by generating 1000 systems of ODEs with each grammar. Node colors correspond to individual nonterminal symbols. Terminal symbols have been omitted to improve readability. The transparency of nodes and edges corresponds to the normalized frequency of the respective derivation paths in collection of parse trees that form the aggregated parse tree. Since the PAGs are composed of too many nonterminal symbols to display in a legend, the legend has been omitted.

by finding all possible closed loops after the circuit topology is generated, the derivation can simply choose and follow one of the predetermined loops whenever the derivation of unknown voltage is required, instead of walking randomly and hoping to be able to close the loop eventually.

Finally, we visualize the space of the derivations, performed by two different PAGs for electronic circuits, in Figure 4.10. The aggregated parse tree on the left-hand side corresponds to a PAG for circuits with four components, and the tree on the right-hand side to a PAG for circuits with eight components. We can see that both aggregated parse trees are substantially more complex than the various aggregated parse trees for PAGs we have seen so far. This reflects the wide variety of the generated circuit topologies, which require different derivations paths through the PAGs. The aggregated parse tree for the 8-component PAG exhibits several extremely long branches. These appear due to long sequences of repeat recursion between two or more unknowns that can lead to endless recursion errors. The recurred sequences can terminate successfully if the derivation of one of the unknowns eventually leads to a known quantity.

The PAG for electronic circuits is not the only option for discovering systems of ODEs that describe electronic circuits. A universal PCFG for mathematical expressions is technically able to generate the equations of electronic circuits. Furthermore, the variables in electronics have distinct dimensions, allowing dimensionally-consistent grammars to impose powerful constraints on the search space. We visualize the search spaces of the three possibilities as AETs in Figure 4.11, addressing a circuit with two capacitors, two inductors and one resistor, which is described by a system of four ODEs. As expected, the

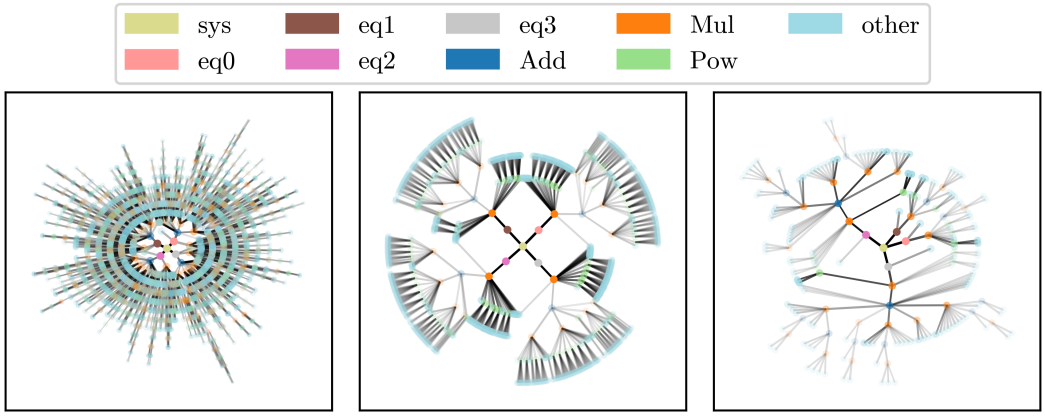


Figure 4.11: Aggregated expression trees of three grammars for electronic circuits: a) a universal mathematical PCFG, b) a dimensionally-consistent universal PAG, c) the PAG for electronic circuits. The AETs were constructed by sampling 1000 random systems of ODEs with each grammar.

universal PCFG encodes a very large space of expressions. The AET of the PAG for electronic circuits reveals a very constrained space, whereas the dimensional PAG still heavily constrains the space, but its AET is substantially larger than the AET of the PAG for electronic circuits. A reasonable approach might make use of both – try to use the PAG for electronic circuits and if it fails, resort to the dimensionally-consistent universal PAG.

The derivations of ODEs for electronic circuits present a serious challenge for most domain knowledge frameworks in equation discovery. We demonstrated one possible way the PAG formalism can encode and apply Kirchoff's laws to generate correct systems of ODEs for a given circuit. Coupled with an algorithm for generating random circuits, such as generate_circuit, this approach can be used to identify unknown RLC circuits. However, issues with dead ends in Kirchoff's closed loops prevent the approach from being able to derive (and consequently discover) the equations of many RLC circuits. As such, without substantial improvements to the PAG that address the use of closed loops, the use of dimensionally consistent PAGs for electronic circuits would likely prove a better choice for the problem of identifying electronic circuits. Nevertheless, even the PAG for electronic circuits in its present form showcases the power and flexibility of the PAG formalism for expressing complex types of background knowledge.

Chapter 5

Bayesian Updating

In the preceding chapters, we developed grammar-based frameworks that focus on constraining the search space and expressing background knowledge for equation discovery. We demonstrated the power of the frameworks in expressing various types of domain knowledge, as well as in enabling a flexible and intuitive parametrization of the parsimony principle. However, as evident from our computational experiments, a significant limitation of the approach is the computationally expensive evaluation of many randomly sampled candidate expressions. The probability of randomly sampling the correct mathematical expression for complex problems is too low for many practical applications of equation discovery, even when leveraging background knowledge. To enable the discovery of more complex equations, an improvement of the algorithm is necessary.

To improve the computational efficiency of probabilistic grammar-based equation discovery, we introduce an algorithm that iteratively updates the probabilities of production rules to guide the search towards more promising areas in the space of mathematical expressions. Production probabilities impose soft constraints on the space of expressions and can be used as parameters of the grammar that the algorithm optimizes. Furthermore, since a PCFG defines a probability distribution over the space of mathematical expressions, this procedure has a Bayesian interpretation in that the resulting distribution is an approximation of the posterior distribution.

In this chapter, we first introduce the algorithm for the iterative updating of grammar probabilities. Then, we demonstrate its behavior on a small, illustrative set of synthetic equation discovery problems.

5.1 m-Estimate Updating Algorithm

We introduce the Bayesian algorithm for updating grammar probabilities by first discussing the estimation of the posterior distribution, which is the core of the method. We start with the simplest approaches for estimating posterior distributions and progressively improve the approximations until we arrive at m-estimate. Next, we apply the m-estimate within the context of updating probabilities of grammar productions. Finally, we present the algorithm as a whole.

5.1.1 m-estimate

Consider a problem with repeated trials, each of which has k possible outcomes. Given evidence with n observations of outcome c in a sample of N trials, we wish to estimate the probability of class c in the next trial. The simplest estimation is relative frequency:

$$p(c) = \frac{n}{N}. \tag{5.1}$$

Relative frequency is a frequently used approximation that estimates the posterior probability based only on the evidence and does not take into account any prior knowledge. A simple way to include prior knowledge into the estimation is Laplace's law of succession. The law assumes that the prior probability of all outcomes is equation – the prior distribution is therefore uniform. Laplace's law of succession [75] estimates the probability of class c as

$$p(c) = \frac{n+1}{N+k}. \tag{5.2}$$

A more general Bayesian method for estimating probabilities allows for more flexibility by using the Beta distribution, parameterized by a and b, as the prior distribution, which estimates the probability of a success in the next trial as [76]

$$q(n, N) = \frac{n+a}{N+a+b}. \tag{5.3}$$

It can be shown that this approximation satisfies the requirements of our problem with k possible outcomes given an appropriate selection of a and b, resulting in the m-estimate [77]:

$$p(c) = \frac{n + p_a(c)m}{N+m}, \tag{5.4}$$

where $p_a(c)$ is the prior probability of outcome c and $m = a + b$ is a parameter of the method.

The parameter m acts as a weight that balances the evidence with the prior probability. For instance, $m = 0$ discards the prior and approximates the posterior as the relative frequency of class c: $p(c) = \frac{n}{N}$. For values of m, much larger than the number of trials N, the evidence is disregarded in favor of the prior: $p(c) = p_a(c)$. By setting m to the number of outcomes ($m = k$) and assuming a uniform prior $p(c)_a = 1/k$, the m-estimate in Equation (5.4) reduces to Laplace's law of succession in Equation (5.2).

The different methods for estimating conditional probabilities have intuitive interpretations. In a problem with two outcomes, such as the problem of determining whether a coin is balanced, Laplace's rule of succession effectively adds two "default" observations to the evidence: one heads and one tails (which follows the uniform prior distribution for two outcomes). This is particularly relevant when the number of trials N is low and the evidence is less reliable.

For integer values of m, the m-estimate generalizes this Laplace's rule of succession by effectively adding m "default" observations that follow the prior distribution to the evidence. This interpretation of the estimation method is useful when choosing the value of m – as the number of "default" observations it can be directly compared to the number of trials in the experiment. Nevertheless, the choice of m is somewhat arbitrary and has to be tuned to the characteristics of a given problem, such as the level of noise in the data [78], [79].

5.1.2 Production rule probability updates

In the context of iterative equation discovery, a trial tests whether a given production rule was used in the derivation of a given expression tree. There are two possible outcomes ($k = 2$) – either the production rule was used or it was not. We compute the posterior by applying the m-estimate to the probability of each production rule in the PCFG.

In order to guide the search towards more promising areas of the search space, we calculate the posterior probability on a subsample of all sampled expressions in an iteration – those with a low error. To that end, we keep track of the lowest error found so far using the algorithm (*best_ error*) and select expressions whose error is lower or within a relative

tolerance ϵ of *best_ error*. In other words, an expression is selected if its *error* fulfills the inequality:

$$\text{error} - \text{best\_error} < \epsilon \cdot \text{best\_error}. \tag{5.5}$$

Alternative selection criteria might consider an absolute tolerance, select a fixed number of expressions with the lowest error, etc. In any case, the selected expressions are considered good examples and are used to estimate the posterior probability of each production rule. By estimating the posterior on a subset of expressions, the estimated posterior is conditioned on the low error of the expressions. Once we have obtained an estimate of the posterior, we update the grammar with the new probabilities. This procedure repeats in each iteration of the Bayesian algorithm.

Algorithm 5.1 details the algorithm for the iterative updating of PCFG probabilities in pseudocode. The procedure is performed in N_{iter} iterations. In each iteration, N_{sample} random expressions are sampled using from the PCFG using Algorithm 1. Each expression is fitted to the data and its error of fit evaluated. Then, following the selection criterion in Equation (5.5), a number of expressions from this iteration are selected, favoring those with low error. Next, the algorithm counts how often each production rule of the grammar appears among the parse trees corresponding to the selected expressions. Then, the procedure updates the probability of each production rule r using the m-estimate from Equation (5.4):

$$p(r) = \frac{n_{\text{prod}}(r) + m \cdot p(r)}{\sum\limits_{r'=A\to\alpha'} n_{\text{prod}}(r') + m},$$

where n_{prod} is the number of occurrences of r in the parse trees of the selected expressions, m is a parameter of m-estimate, $p(r)$ is the current (prior) probability of r and $\sum\limits_{r'=A\to\alpha'} n_{\text{prod}}(r')$ is the total number of occurrences of each production rule with the same left-hand side as r in the selected parse trees.

Once the probabilities of all production rules are updated, the algorithm proceeds to the next iteration. The result of the procedure are the final probabilities of production rules, as well as a list of mathematical expressions, each with its corresponding error-of-fit. Finally, an approximation of the posterior probability of each expression may be obtained by parsing the expression using the PCFG with the final values of its probabilities.

5.2 Empirical Evaluation

We study the performance of the proposed Bayesian grammar updating algorithm with a small, demonstrative empirical experiment.

5.2.1 Experimental setup

The experiment tests the ability of the algorithm to discover the following three equations:

$$y = x_1 - 3x_2 - x_3 - x_5, \tag{5.6}$$

$$y = x_1^5 x_2^3, \tag{5.7}$$

$$y = \sin(x_1) + \sin\left(\frac{x_2}{x_1^2}\right). \tag{5.8}$$

Algorithm 5.1: DISCOVER_EQUATIONS_BAYESIAN
Bayesian updating of grammar probabilities based on the m-estimate.

Data: Probabilistic grammar $G = (\mathcal{N}, \mathcal{T}, \mathcal{R}, S)$ generating mathematical expressions, data set D, target variable v, number of iterations N_{iter}, number of samples per iteration N_{sample}, m-estimate parameter m, error tolerance ϵ.

Result: Equation with lowest error, list of all evaluated equations *eqns*.

initialize *evaluated_eqns* = [ ];
initialize *best_error* = inf;
for $n = 1$, $n \leq$ *max_iter* **do**
  randomly sample N_{sample} *expressions and evaluate them*;
  initialize *eqns* = [ ];
  for $i = 1$, $i \leq N_{sample}$ **do**
    (e, p) = GENERATE_SAMPLE(G, S);
    e_c = CANONICAL_FORM(e);
    eqn = FIT_PARAMETERS(e_c, v, D);
    error = RMSE(*eqn*, D);
    if *error* < *best_error* **then**
      best_error = *error*;
      best_eqn = *eqn*;
    end
    eqns.append((*eqn*, *error*));
  end
  count the occurences of each production rule in each selected parse tree;
  initialize n_{prod};
  for *(eqn, error) in eqns* **do**
    if *error* − *best_error* < $\epsilon \cdot$ *best_error* **then**
      n_{prod}.update(*eqn*)
    end
  end
  update grammar probabilities using m-estimate;
  for *production rule* $r = A \to \alpha \in \mathcal{R}$ **do**
    $p_{\text{new}}(r) = \frac{n_{\text{prod}}(r) + m \cdot p(r)}{\sum_{r' = A \to \alpha'} n_{\text{prod}}(r') + m}$
  end
end
return the expression with the lowest error, as well as all the evaluated expressions
return (*best_error*, *best_eqn*, *evaluated_eqns*);

For each of the three equations, we generate 100 data points by uniformly sampling the variables in the $(-10, 10)$ interval. We use a variant of the universal grammar for mathematical expressions with the following initial production rule probabilities:

$$\begin{aligned} &E \to E + F \ [0.2] \mid E - F \ [0.2] \mid F \ [0.6] \\ &F \to F * T \ [0.2] \mid F \ / \ T \ [0.2] \mid T \ [0.6] \\ &T \to (E) \ [0.2] \mid \sin(E) \ [0.2] \mid V \ [0.6] \\ &V \to x_1 \ [1/n_v] \mid x_2 \ [1/n_v] \mid \ \ldots \ \mid x_{n_v} \ [1/n_v], \end{aligned} \tag{5.9}$$

where n_v is the number of variables. For simplicity, the grammar does not generate numerical parameters and the three equations do not feature real constants.

Each of the three equations presents a challenge for most equation discovery approaches, but particularly for Monte-Carlo sampling of PCFGs. The dataset for the first equation features five variables, whereas the equation itself uses only four of the variables. Discovering an expression of five variables is very difficult by randomly sampling a PCFG. Furthermore, the grammar does not include numerical constants, requiring repeated addition or subtraction to generate the integer constants in the expression. As such, although the expression is simple, featuring only addition and subtraction, its probability of generation with the PCFG is incredibly low.

The second equation focuses on multiplication. It is composed of only two variables, but with relatively high exponents. Since we are using a grammar that does not explicitly include the power function, the algorithm must rely on repeated multiplication to generate the expression. Due to the high values of the exponents, this expression is also highly unlikely to be generated using the grammar.

Both the first and the second expression are designed to encourage a clear optimization path for the grammar probabilities: for the first equation, by favoring addition and subtraction and ignoring the missing variable, and for the second equation, by ignoring all operations except for multiplication. In contrast, the third equation provides no obvious optimization path. It uses almost all of the mathematical operations in the grammar, including division, and further complicates the task through a trigonometric function. This equation is included in the set to test how the algorithm performs in a clearly disadvantageous scenario.

In the experiment, we compare four versions of the Bayesian algorithm based on m-estimate updates (MEU) with varying values of m: 1, 2, 5, 10, and the baseline Monte-Carlo algorithm from Chapter 2, where we keep production probabilities constant.

We give each tested method a budget of 100000 expressions to evaluate for each of the three problems. In the case of the Monte-Carlo algorithm, this simply means sampling 100000 random expressions with the grammar and computing their errors. We run all four versions of MAE in 2000 iterations, sampling 50 random expressions in each iteration. If no expression in an iteration passes the relative threshold selection criterion, we do not update probabilities and move to the next iteration. In the experiment, we use $\epsilon = 0.1$ as the relative threshold. This selects expressions with an error lower than the lowest error so far, or within 10% of the lowest error. To account for the randomness involved in the algorithms, we run each of the five evaluated algorithms 10 times with different random seeds.

5.2.2 Results: The error-of-fit

We summarize the results in Table 5.1 by considering how often each equation was discovered exactly among 10 runs of each compared method. The first equation proved the most difficult, as it was discovered only in a single run of the Bayesian updating for $m = 1$. The contrast between the Bayesian updating and the Monte-Carlo algorithms was most prominent for the second equation – the Bayesian algorithm was able to discover it in all but one run, while Monte-Carlo discovered it only in one out of ten runs. The third equation also proved troublesome for most algorithms.

To further compare the different methods, we calculate and plot optimization curves by finding the mean value of the lowest RMSE error across the ten runs at each iteration of the procedure. We compare the optimization curves of the four Bayesian methods and the Monte-Carlo approach for each of the three equations in Figure 5.1. The plots mirror the findings from Table 5.1, with the mean error of Bayesian approaches dropping to

Table 5.1: The number of equation discovery successes (exactly recovered equation) among 10 runs with different random seeds for the four variants of Bayesian m-estimate updating and the Monte-Carlo sampling algorithm (random).

	$m = 1$	$m = 2$	$m = 5$	$m = 10$	random
$y = x_1 - 3x_2 - x_3 - x_5$	1	0	0	0	0
$y = x_1^5 x_2^3$	10	10	10	9	1
$y = \sin(x_1) + \sin\left(\frac{x_2}{x_1^2}\right)$	0	1	0	3	1

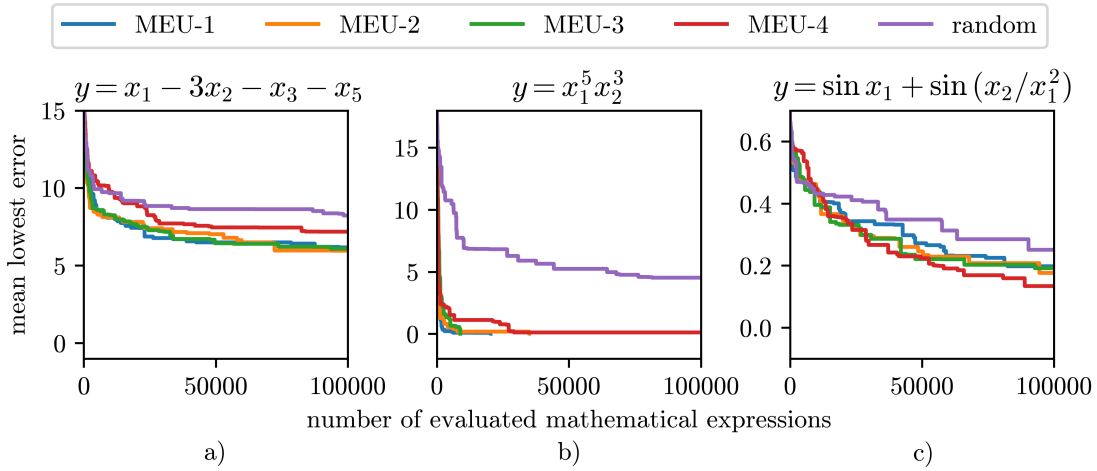


Figure 5.1: Optimization curves of the Bayesian m-estimate updating algorithm (MEU-m, indicating the value of the parameter m) and the Monte-Carlo sampling algorithm (random) for each of the three equations: a) $y = x_1 - 3x_2 - x_3 - x_5$, b) $y = x_1^5 x_2^3$, c) $y = \sin x_1 + \sin(x_2/x_1^2)$. The horizontal axis depicts the total number of evaluated expressions, whereas the vertical axis depicts the lowest error (RMSE) achieved for a given number of expressions, averaged across 10 runs with different random seeds.

zero quickly for the second equation and remaining relatively high for the first and third equation. The most important observation to make, however, is that the mean error of Bayesian approaches is lower than the mean error of the Monte-Carlo approach for almost the entire length of the optimization procedure for all three equations. In other words, the Bayesian approach clearly outperformed random sampling in this experiment. The choice of m did not prove to have a notable impact on the performance of the Bayesian updating algorithm, with the optimization curves for all four options being relatively close. We chose $m = 2$ for further analysis as the option with the most consistent results across the three equations in the experiment.

5.2.3 Results: Production rule probabilities

We can further investigate the behavior of the Bayesian grammar updating algorithm by studying how the probabilities of production rules evolve throughout the optimization procedure. For each of the three equations in the experiment, we choose one of the $m = 2$ runs with the lowest error for analysis. Figure 5.2 depicts the probabilities of production rules at each iteration for the first equation. The first of the four plots compares the probabilities of production rules with the nonterminal E on the left-hand side. We can see

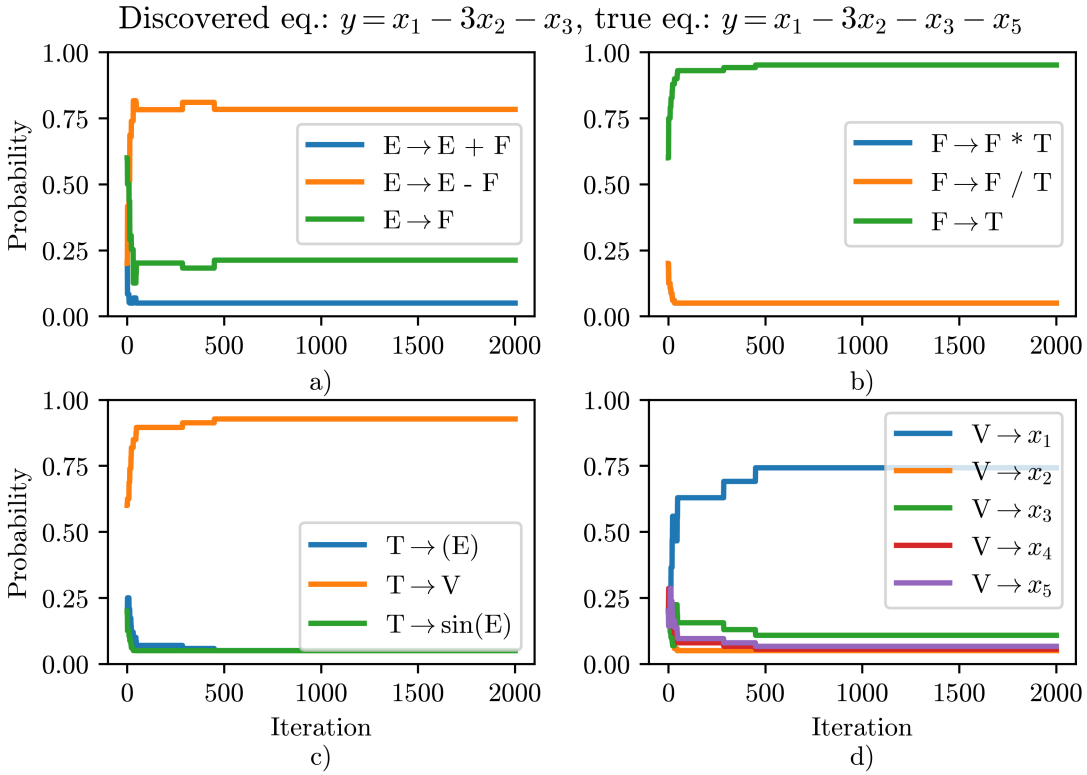


Figure 5.2: The probabilities of production rules with the nonterminal a) E, b) F, c) T, d) V on the left-hand side, plotted at each iteration of the Bayesian grammar updating algorithm ($m = 2, run = 7$) for the first equation in the experiment. In this run, the algorithm discovered an approximation of the target equation, which misses only the term $-x_5$, and achieves the error RMSE = 5.99.

that the algorithm very quickly learned a strong preference for subtraction over addition. This is expected since generating the target equation requires five instances of subtraction and only one instance of addition (since there are no numerical constants in the grammar, the simplest way to generate $-3x_2$ is as $-x_2 - x_2 - x_2$). In the second plot in the first row, we observe that the algorithm quickly learned to completely ignore both multiplication and division, which also follows our expectation. Next, the first plot in the second row indicates a reasonable disregard for the sine function. Interestingly, the probability of the complicated recursion (E) also tends towards zero. Although this production rule is not as obviously counterproductive as the sine function or multiplication and division, it significantly increases the complexity of generated equations, which is not needed for the target equation. The final plot reveals that the algorithm learned to prefer the variable x_1 over all other variables. This is the first observation that does not conform to our expectations. Due to the absence of numerical constant, we would expect a preference for x_2, since it must be generated three times, compared to only once for each other variable. As the data was sampled from identical distributions for all five variables, the effect can also not be attributed to a higher importance of x_1 in the data. We therefore conclude that the algorithm followed a sub-optimal optimization path in the distribution of variables, which may have contributed to its failure in exactly recovering the target equation.

Figure 5.3 depicts the evolution of production rules throughout the updating procedure for the second equation in the experiment. This equation features no addition, subtraction,

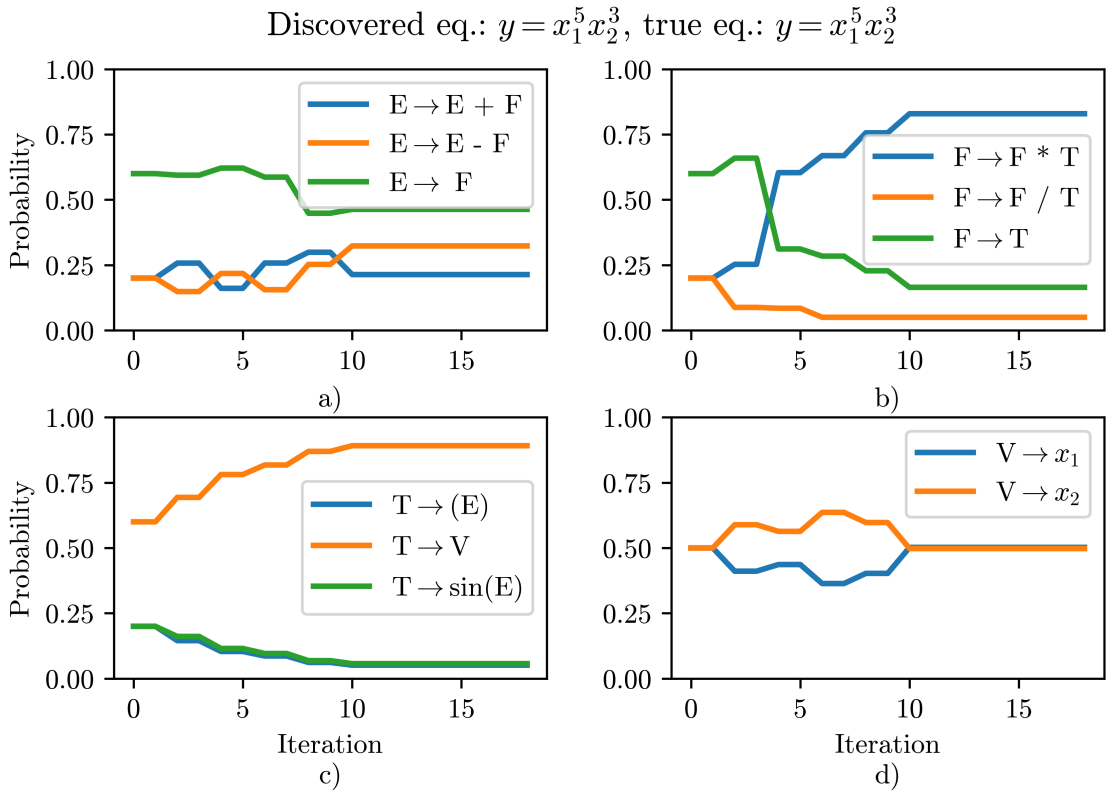


Figure 5.3: The probabilities of production rules with the nonterminal a) E, b) F, c) T, d) V on the left-hand side, plotted at each iteration of the Bayesian grammar updating algorithm ($m = 2, run = 6$) for the second equation in the experiment. In this run, the algorithm was able to exactly recover the target equation with an error of RMSE = 0.

division or special functions, but requires a lot of multiplication. We also expect x_1 to be preferred over x_2. The first plot reveals that, contrary to our expectations, the algorithm did not learn to ignore addition and subtraction, but even increased their probability slightly. On the other hand, our expectations were met in the second set of production rules, where the probability of multiplication rises over 0.8 and division is suppressed entirely. Very similarly to the behavior for the first equation, the probability of (E) and sine dropped to zero. Finally, the algorithm once again failed to learn any variable preferences. However, note that the optimization procedure for the second equation terminated extremely early, as an equation with $RMSE = 0$ was discovered in the 19th iteration. As such, it is likely that not all probability distributions have not had time to converge. All in all, the behavior of the algorithm when discovering the second equation makes sense, as it successfully learned the crucial importance of multiplication.

We expected the third equation in the experiment to be the most difficult, as it features complicated nested functions and a trigonometric function with a varying frequency. Nonetheless, the Bayesian algorithm with $m = 1$ was able to discover it exactly in one of the runs, depicted in Figure 5.4. In the first plot we observe a clear preference of addition over subtraction, which is expected, since the equation features addition once and does not feature subtraction. Very similarly, the algorithm successfully learned it does not need multiplication, but that division is important. The third plot is highly interesting, as it shows the same behavior as for the first two equations – dropping the probability of (E)

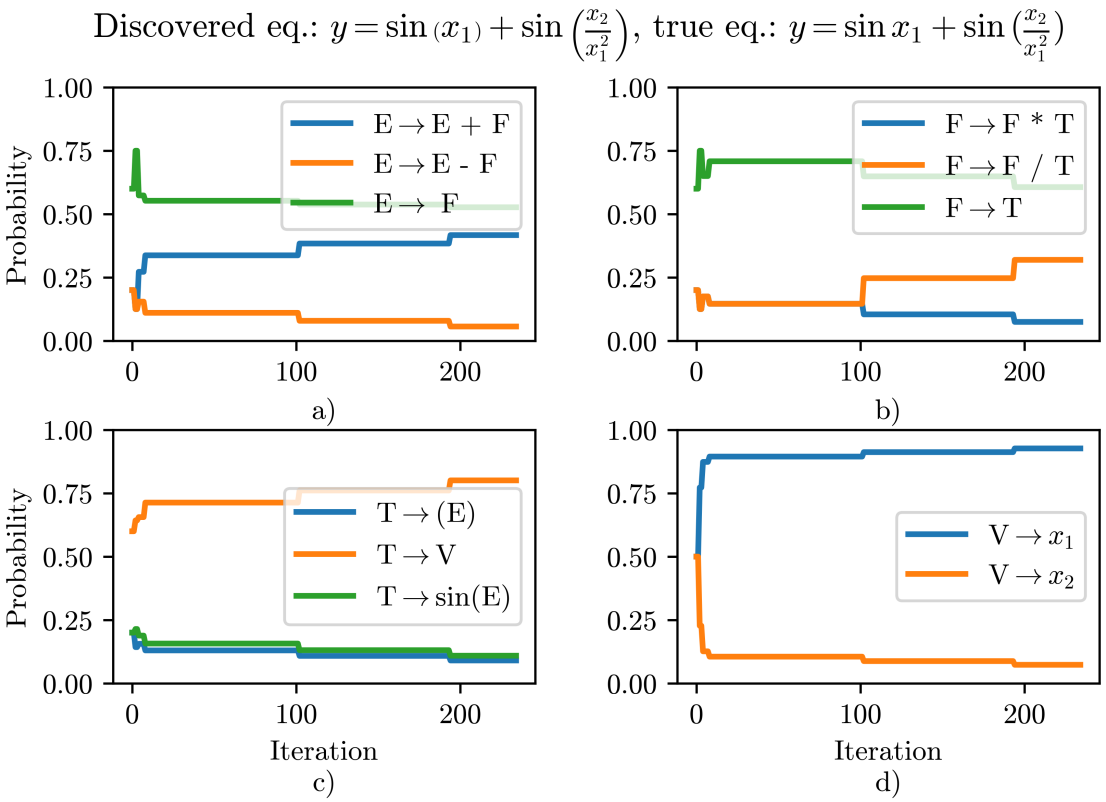


Figure 5.4: The probabilities of production rules with the nonterminal a) E, b) F, c) T, d) V on the left-hand side, plotted at each iteration of the Bayesian grammar updating algorithm ($m = 2, run = 5$) for the second equation in the experiment. In this run, the algorithm was able to exactly recover the target equation with an error of RMSE = 0.

and sine. This is surprising, because these two production rules are crucial for deriving the third equation. In contrast to the probabilities for the first two equations, here, the probabilities do not drop to zero, but stay at approximately 0.1. This may indicate that the initial grammar probabilities generated too complex expressions and more parsimony was required. In the fourth plot, we finally observe a case where the algorithm learned a preference of variables. This preference, however, is extreme, with $p(x_1) > 0.9$ and $p(x_2) < 0.1$, where we would expect this ratio to be around 3:1 in favor of x_1. Nevertheless, the overall behavior of the algorithm in the case of the third equation is well in line with our intuition.

5.2.4 Results: Posterior probabilities

One of the advantages of the Bayesian grammar updating algorithm is that it approximates the posterior distribution over the space of mathematical expressions. Once we have completed the optimization of grammar probabilities, we can calculate the posterior probability of individual expressions by parsing them using the PCFG with updated probabilities. Note, however, that because each mathematical expression can be derived by the grammar in a number of different ways, the probability obtained this way is only an approximation of the true probability. On the other hand, this is not an issue when comparing expression probabilities, obtained by parsing the expression in the same way using the same grammar, with different values of production rule probabilities, such as when comparing the prior and posterior expression probabilities. To further understand

the behavior of the Bayesian grammar updating algorithm, we compute the probability of the correct expression at various points in the procedure, including the initial and final iteration. To obtain more accurate approximations of the probability, we rewrite the first equation from the experiment following the commutativity of summation by generating every possible ordering of the terms:

$$y = x_1 - 3x_2 - x_3 - x_5 = x_1 - x_2 - x_2 - x_2 - x_3 - x_5 = -x_2 + x_1 - x_2 - x_2 - x_3 - x_5 = \dots$$

In total, we generate 120 different expressions, parse each using the initial PCFG and the final PCFG to obtain its prior and posterior probabilities, respectively, and sum the individual probabilities to obtain an approximation of the prior and posterior probabilities of the correct expression. We treat the second equation from the experiment in a similar way, generating 56 orderings of the factors:

$$y = x_1^5 x_2^3 = x_1 * x_1 * x_1 * x_1 * x_1 * x_2 * x_2 * x_2 = x_2 * x_1 * x_1 * x_1 * x_1 * x_1 * x_2 * x_2 = \dots$$

Finally, the third equation has only four simple ways of rewriting it for the grammar we used in the experiment:

$$\begin{aligned} y = \sin(x_1) + \sin\left(\frac{x_2}{x_1^2}\right) &= \sin(x_1) + \sin\left(x_2/(x_1 * x_1)\right) = \sin\left(x_2/(x_1 * x_1)\right) + \sin(x_1) = \\ &= \sin(x_1) + \sin\left(x_2/x_1/x_1\right) = \sin\left(x_2/x_1/x_1\right) + \sin(x_1). \end{aligned} \tag{5.10}$$

In Table 5.2, we summarize the prior and posterior probabilities for each of the three equations using the final PCFGs from Figures 5.2-5.4.

Table 5.2: Approximated prior and posterior probabilities of the correct expression for each of the three equations from the experiment, obtained by parsing the many equivalent mathematical expressions using the PCFG with the initial and final values of production rule probabilities.

	Equation 1	**Equation 2**	**Equation 3**
Prior probab.	$5.3 \cdot 10^{-10}$	$1.7 \cdot 10^{-8}$	$1.6 \cdot 10^{-7}$
Posterior probab.	$1.7 \cdot 10^{-7}$	$1.3 \cdot 10^{-3}$	$4.8 \cdot 10^{-7}$

The posterior probability of the correct expression is higher than the prior probability for all three equations, proving that the Bayesian algorithm indeed leads the distribution towards the correct expression. The improvement is the lowest for the third equation, with the posterior being three times the prior, and the highest for the second equation, improving the probability by 5 orders of magnitude.

Note that we performed this analysis on the most successful runs in the experiment. In order for the Bayesian algorithm to be really useful, the posterior probability should be improved even in cases when it does not succeed in discovering the correct expression. To test this property, we perform the above analysis for every run from the experiment, using PCFGs throughout the optimization procedure. We plot the evolution of the minimum, median and maximum probability across the ten runs in Figure 5.5. We see that in the median probability of the correct expression does indeed increase for all three equations, although it experiences rather erratic jumps during the optimization process. The lowest final probability for the first equation is still an order of magnitude higher than the initial. For the second equation, the lowest final probability is the same as the initial probability, and lower than the initial probability for the third equation. We can therefore conclude that

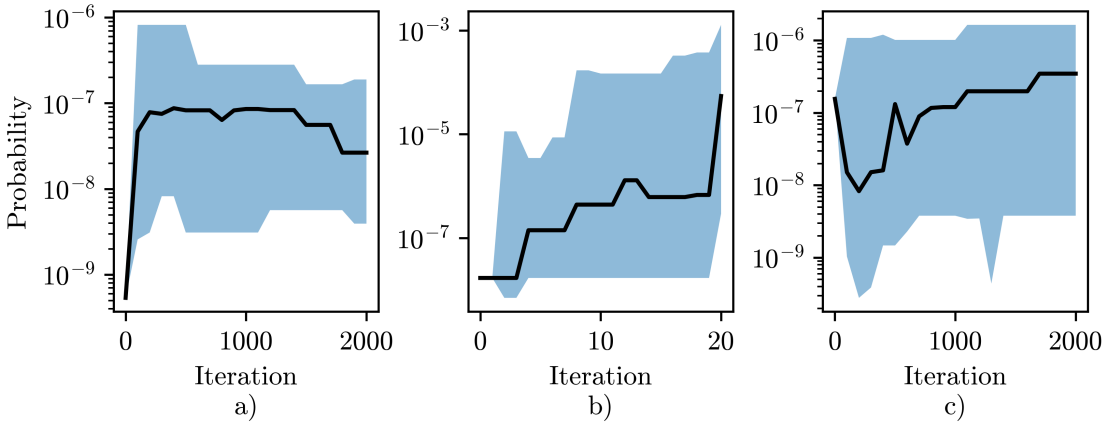


Figure 5.5: The approximated probability of the correct expression at each iteration of the Bayesian grammar updating algorithm ($m = 2$) for each of the three equations: a) $y = x_1 - 3x_2 - x_3 - x_5$, b) $y = x_1^5 x_2^3$, c) $y = \sin x_1 + \sin(x_2/x_1^2)$. The black line depicts the median probability and the blue area depicts the region between the minimum and maximum probability among the 10 runs.

in an average run of this experiment, the Bayesian algorithm will result in an increased probability of the correct expression. Runs in which the Bayesian algorithm achieves a counterproductive effect, although unlikely, are a very real possibility.

5.2.5 Results: Aggregated expression trees

Throughout the thesis, we have been studying the space of mathematical expressions, defined by a grammar, with the help of the visualizations of aggregated parse trees. We again make use of them to visualize how the space of expressions evolves during the grammar updating procedure in Figure 5.6.

For all three equations, the space of expressions shrinks visibly, which is further confirmed by the number of nodes in each aggregated parse tree. The reduction in the size of the space is the most significant for the first equation, with the number of nodes in the final AETs dropping to a tenth of the number in the initial AET. For the second and third equation, the number of nodes drops to about a third of the number in the initial AETs. This confirms that the Bayesian algorithm guides the search from a highly uninformative prior, encoding a very unconstrained space of expressions, towards a biased posterior, encoding a progressively more constrained space of expressions. This finding corresponds with the observation of the increasing posterior probability of the correct expression.

In summary, the performance of the Bayesian grammar updating algorithm in exactly reconstructing the target equations in our limited experiment is not impressive, since it was able to discover the first and third equation in only a small number of runs. On the other hand, it was very successful in discovering the second equation. Furthermore, the optimization curves reveal that on average, the Bayesian algorithm clearly outperforms the previous Monte-Carlo sampling approach. Most importantly, the analysis of the evolution of production rule probabilities and the posterior probability of the correct expressions shows that the algorithm is indeed able to guide the search from highly uninformative prior distributions towards biased distributions that exhibit the expected properties of target equations.

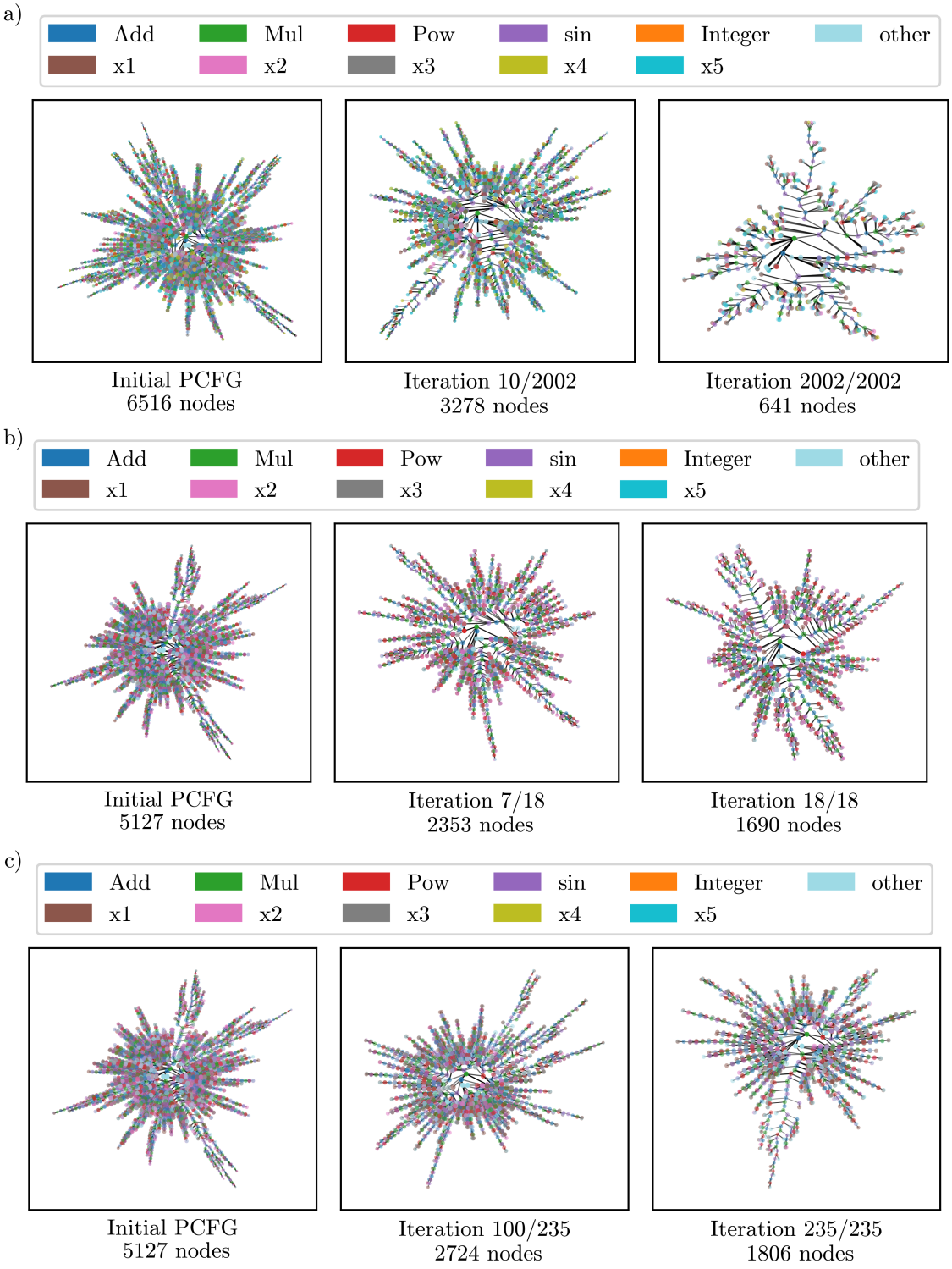


Figure 5.6: Aggregated parse trees, depicting the evolution of the space of expressions, defined by the initial PCFG, the PCFG at an intermediate point of the Bayesian grammar updating procedure (the most successful runs for $m = 2$) and the final PCFG. Row a) corresponds to the target equation $y = x_1 - 3x_2 - x_3 - x_5$, row b) to the equation $y = x_1^5 x_2^3$ row c) to the equation $y = \sin(x_1) + \sin\left(\frac{x_2}{x_1^2}\right)$.

5.3 Computational Efficiency and Parallelization

Our primary purpose in developing the new Bayesian updating method is to improve the performance and computational efficiency of equation discovery by reducing the number of candidate expressions that need to be evaluated. In that respect, the new algorithm shows promise, since it on average achieves a lower error with the same number of evaluated expressions than the Monte-Carlo algorithm (as shown in Figure 5.1). The overhead, introduced by counting production rules and updating probabilities is insignificant compared to the effort of evaluating candidate expressions, particularly when parameter estimation is required.

However, the Bayesian algorithm is not without downsides, the most concerning of which is its difficulty of parallelization. Heavy parallelization of the expression evaluation step is the key to achieving reasonable computation times for the Monte-Carlo method. In fact, parallelizing the Monte-Carlo method is very simple, since expressions are sampled completely independently from each other. In our computational experiments in Chapters 2 and 3 we generated a large number of candidate expressions locally, which was a very fast process. We then made use of supercomputing clusters to perform the parameter estimation and evaluation of as many as 1000 generated expressions in parallel.

In contrast, the Bayesian grammar updating algorithm proceeds in iterations and each iteration depends on the results of the previous iteration. This means that it is much more difficult to evaluate the generated expressions in parallel. Parallelization is possible in two levels: either by executing several runs (i.e., using different random seeds) of the algorithm in parallel, or by evaluating the limited number of candidate expressions within an iteration in parallel. The latter option requires a complicated parallelization scheme, where a main thread must wait for all the parallel evaluation threads to finish, compile their results and update the probabilities, generate the candidates for the next iteration, before starting a new parallelized evaluation step for the next iteration. This scheme is possible and would still bring significant performance increases, but it is more difficult to implement and brings limited benefit compared to the full parallelization of the Monte-Carlo approach. Due to the efficiency of Monte-Carlo algorithm parallelization, when powerful HPC resources are available, iterative grammar sampling approaches must achieve a significant performance advantage to outperform Monte-Carlo sampling in practical settings.

The development of a more sophisticated and efficient algorithm for generating expressions from grammars is crucial for the practical applicability of the grammar-based approaches introduced in this thesis. In this chapter, we introduced one possible way to address this problem, based on an iterative, Bayesian updating of PCFG probabilities. The results demonstrate that the method works and outperforms the previous Monte-Carlo approach when comparing the number of evaluated candidate equations. The probabilities of the grammar mostly evolve in an expected way and guide the search towards areas of the expression space that exhibit the properties of the target equations. An additional benefit of the method is that it approximates the posterior distribution to the Bayesian probabilistic nature of the approach, which is beneficial to the interpretability of equation discovery.

On the other hand, the method leaves a lot of room for improvement. In a minor, but non-negligible number of runs, the method was not only unable to discover the correct equation, but also resulted in a PCFG with a lower probability of the correct equation than the initial PCFG, meaning that the algorithm guided the search in the wrong direction. Furthermore, the iterative nature of the algorithm makes the approach difficult to parallelize, which can outweigh any gains in computational efficiency when considerable computational resources are available.

Chapter 6

Conclusions

In the introduction to this thesis, we outlined its purposes, broke down the steps required to fulfill the purposes, and most importantly, we defined the research hypotheses to be answered. In this final chapter, we review the accomplishments in this work, evaluate its contributions and answer the scientific questions raised in the introduction.

6.1 Summary

In this thesis, we address various challenges in the field of equation discovery through the use of probabilistic grammars. Before evaluating our accomplishments, we review the presented work.

6.1.1 Probabilistic context-free grammars

In the second chapter of this thesis, we introduced a method for equation discovery, based on randomly sampling probabilistic context-free grammars. We constructed a theoretical framework for the use of PCFGs as generators of mathematical expressions. The first component of the framework consists of an algorithm for counting the number of parse trees up to a given height in a context-free grammar and an algorithm for computing the total probability of generating a parse tree up to a given height using a PCFG. We employed this component to demonstrate how PCFGs naturally encode and parametrize the parsimony principle. The second component leverages the functionality of parsing, enabled by the use of PCFGs, to compare and estimate the performance of different grammars in the task of equation discovery. We demonstrated this by comparing the expected number of expressions required to solve each problem from the Feynman database using a deterministic and a probabilistic universal grammar for mathematical expressions, as well as a uniform and a biased version of the universal PCFG for mathematical expressions.

To enable probabilistic grammar-based equation discovery, we developed a simple algorithm (Algorithm 1) that randomly generates mathematical expressions from a PCFG. Besides its direct application in equation discovery, the algorithm enabled the development of an empirical framework for analyzing PCFGs in the context of equation discovery, which complements the theoretical framework. Using a large sample of randomly generated expressions, the first component of the empirical framework allows us to visualize the search space, defined by a PCFG, using an aggregated expression tree. More complex metrics, computed on the aggregated expression tree, summarize high-level properties of the grammar, such as the level of bias, i.e., the balance of exploration and exploitation, exhibited by the grammar. After fitting the parameters of each generated expression and computing its error-of-fit, the second component of the empirical framework enables the estimation

of a learning curve, obtained by repeated bootstrap resampling of the sampled expressions and their respective errors. This learning curve provides us with an estimate of performance (i.e., the probability of successfully discovering an equation) for any given number of randomly sampled expressions, up to the sample size used in the actual sampling.

The theoretical and empirical analyses reveal that the use of PCFGs offers a versatile and potent framework for defining the inductive bias in the context of equation discovery and symbolic regression. Specifically, probabilistic grammars enable a concise definition of the prior distribution across the pool of candidate equations. This is accomplished by assigning probability distributions to the production rules associated with each non-terminal symbol within the grammar. For instance, this grants the user of the equation discovery algorithm intuitive and transparent control over the principle of parsimony by stipulating the probabilities of recursive production rules that govern the generation of mathematical expressions. Reducing the probability of these recursion rules increases the likelihood of simpler equations.

This outcome holds significant importance, given that the parsimony principle, which aligns with Occam's razor by favoring simpler explanations over complex ones, plays a pivotal role in algorithms designed for equation discovery and symbolic regression [13]. Historically, addressing this principle has entailed a variety of techniques, such as the minimum-description-length formalism [7], [8], [13], [48], Akaike and Bayesian information criteria [8], regularization methods applied to sparse regression [3], and the inclusion of complexity-related penalty terms in the fitness functions used in evolutionary approaches [6]. These approaches to encoding the parsimony principle often necessitate the careful tuning of a regularization parameter, which determines the trade-off between equation error and complexity. Discovering the optimal parameter configuration can involve computationally intensive trial-and-error experiments. In contrast, the framework outlined here establishes a foundation for analytically determining the probabilities of recursive rules based on the probabilities associated with simpler expressions, corresponding to shallower parse trees.

Incorporating domain-specific background knowledge into the equation discovery process bears a strong connection to the communicability of the resulting equations to experts from the relevant fields. Equations and mathematical models aligned with the domain's existing knowledge enable humans to interpret them as explanations for observed phenomena [12]. Deterministic grammars, along with other constraint types, as outlined in [11], serve as mechanisms for knowledge integration, but in doing so, exclude entire regions from the pool of candidate equations. While these exclusions enhance computational efficiency, they also introduce the risk of discarding potentially valid models. Probabilistic grammars, on the other hand, offer a more flexible approach through soft constraints, thereby mitigating this risk by specifying a probability distribution across the space of candidate equations. Simultaneously, probabilistic grammars can enhance the computational efficiency of the equation discovery process and yield models that are easily communicable. The parse trees generated by the grammar, along with their internal nodes, correspond to non-terminals and potentially explanatory higher-order expressions, much like the processes found in explanatory process-based models [26].

We evaluated the performance and the computational efficiency of the approach, based on randomly sampling a universal PCFG for mathematical expressions, through an extensive computational experiment using the Feynman database for symbolic regression, composed of a selection of one hundred of the most important equations from physics. Our PCFG-based approach was able to discover 36 – significantly fewer than competing methods, such as AI Feynman [13]. In particular, our method was able to discover simpler equations, while failing to discover more complex ones. This result is not surprising, since we employed a very simple equation discovery algorithm that relied on randomly

sampling mathematical expressions from a PCFG. Furthermore, we used a universal grammar for mathematical expressions for our experiments, which implies the absence of any background knowledge. Since the strength of grammars lies in their ability to efficiently express background knowledge, the use of problem-specific knowledge, coupled with a more powerful search algorithm, is required to achieve a competitive performance of equation discovery.

6.1.2 Dimensionally-consistent equation discovery

In the next chapter of this work, we focused on a particular type of background knowledge – dimensional analysis. Measurement units of relevant variables are frequently known in physical and engineering problems. Methods of dimensional analysis, formalized in the Buckingham Π theorem [45], can often significantly simplify a problem and sometimes outright solve it. In our work, we leveraged the knowledge of measurement units to generate only dimensionally-consistent mathematical expressions, potentially significantly constraining the search space. To that end, we introduced attribute grammars as a framework that combines the ability of context-free grammars to express different types of domain knowledge with dimensional analysis, a more restrictive form of background knowledge.

In attribute grammars for dimensionally-consistent expressions, each nonterminal symbol has an attribute representing its measurement unit. Production rules are equipped with attribute rules that encode the rules for measurement unit arithmetic and constrain the selection of variables based on measurement units. In this way, dimensional consistency is guaranteed entirely by attributes and attribute rules, keeping the production rules of the grammar simple and available for the encoding of other background knowledge available in a given problem.

In this first formulation of attribute grammars for equation discovery, specialized for dimensional analysis, we relied on transforming the attribute grammar to a PCFG, which allowed us to use the existing algorithm for sampling PCFGs to generate dimensionally-consistent expressions. The algorithm for transforming an attribute grammar to a PCFG involves enumerating the possible values of attributes (measurement units) and introducing new nonterminal symbols for each relevant combination of the nonterminal symbol and attribute value (measurement unit).

We found two issues with this approach. Firstly, the PCFGs obtained this way are large, composed of many nonterminals and production rules, which makes them difficult to understand and interpret through manual inspection. Secondly, for many problems, it is difficult to select an appropriate set of measurement units to include. Since the grammar derives mathematical expressions step by step, the generation process can fail if an intermediate measurement unit, required for the derivation of a unit on the left-hand side of the equation, is missing from the selected set of units. We were able to solve this issue by developing a heuristic algorithm that identifies a set of measurement units that works, but is not necessarily minimal. A downside of the solution is that the sampling algorithm can frequently encounter dead ends in the derivation, which requires many repetitions of the sampling process.

Finally, we evaluated the performance of using grammars as generators of dimensionally-consistent expressions in equation discovery through a computational experiment using the Feynman database. We found that ensuring dimensional consistency significantly improves the performance of ProGED on the benchmark database, increasing the number of successfully discovered equations from 36 to 58. Furthermore, dimensional consistency significantly improves the computational efficiency of equation discovery by reducing the number of physically impossible mathematical expressions, thereby allowing the algorithm to find a solution faster.

6.1.3 Probabilistic attribute grammars

Transforming a probabilistic attribute grammar into a PCFG is possible only for problems with a small number of attributes, which have a limited number of possible attribute values. To enable more universal applications of PAGs in equation discovery, we designed a new framework for expressing background knowledge with PAGs, coupled with a direct sampling algorithm for PAGs.

In the new framework, attribute rules are expressed as Python code and attributes are arbitrary Python objects. Attribute rules serve to propagate attributes upward or downwards in a parse tree, to restrict the selection of production rules based on attribute values, and to impose conditions on the attribute values of nonterminals, derived in diverging branches of a parse tree. In other words, attributes and attribute rules enable a limited form of context sensitivity in probabilistic grammars, thereby allowing the framework to express many types of background knowledge that were difficult or impossible to express using PCFGs only.

We demonstrate the use and utility of the PAG framework by designing PAGs for three different types of background knowledge. First, we return to the problem of dimensional consistency, addressed in Chapter 3, and solve it using the new framework. The direct sampling algorithm simplifies the design and use of dimensionally-consistent grammars. On the other hand, the sampling process is significantly slower for PAGs than for dimensionally-consistent PCFGs, due to the cost of executing and evaluating the attribute rules of each production rule.

The second type of background knowledge we address are dynamical systems. The primary challenges with dynamical systems are related to the fact that dynamical systems are often represented by a system of ordinary differential equations. A context-free grammar can generate a system of equations by including a separator, such as a colon, among its terminal symbols and processing the generated string accordingly. However, domain knowledge associated with systems of coupled equations can be complicated and impossible to express without context sensitivity. For instance, many dynamical systems feature so-called coupling terms that appear in several equations in the system as the same structure. Using our framework, we design a grammar for dynamical systems that can ensure the identity of coupling terms in a system of ODEs.

Next, we focus on a particular type of dynamical systems that frequently appears in the field of chemical kinetics. As a type of population model that describes the time evolution of the concentrations of chemical reactants and products, the governing system of ODEs is highly restricted. To demonstrate the use of our PAG framework, we explicitly identify the rules that constrain the mathematical expressions composing a system of ODEs for chemical kinetics. We then design a PAG that generates only systems of ODEs following the restrictions of chemical kinetics.

As the final domain knowledge example, we tackle the problem of modeling electronic circuits. These also represent a type of dynamical systems, but the associated background knowledge is remarkably complex. We focus on so-called RLC circuits – analog electronic circuits, composed of resistors, inductors and capacitors, as well as voltage sources. Each component is connected to two other components by wire. A wire can also split into any number of wires through a junction, but the entire circuit must not have any open connections. As such, the topology of a circuit is as important to its function as its electronic components. If the components and topology of a circuit are known (i.e., we can draw a circuit diagram), the governing system of ODEs can be derived by the application of Kirchoff's laws.

Designing an equation discovery tool that expresses this domain knowledge and can generate physically-correct systems of ODEs is a very difficult task, beyond the reach of

context-free grammars and even process-based models. As a final test of the expressive power of our attribute grammar framework, we develop a PAG that generates systems of ODEs that obey the physics of RLC circuits. The PAG uses attributes to embed the topology of a circuit, as well as to help control the flow of the derivation. We developed a separate algorithm that generates a random circuit topology, i.e., it randomly connects a given list of electronic components into a valid circuit. Sampling the PAG results in a single valid system of ODEs – the correct one. The production rules of the PAG express both Kirchoff's laws and allow for the recursive derivation of each unknown quantity.

The PAG for electronic circuits is an attempt to push the expressive power of the PAG framework to its limits. It demonstrates how PAGs can be used to encode a famously troublesome type of domain knowledge and identify unknown circuits from data. However, this attempt is only partially successful. Namely, the derivation of the system of ODEs fails for many given random circuits due to dead ends during the traversal of closed loops and due to endless recursion that results from the direct application of Kirchoff's laws. To make things worse, the fail rate increases with increasing circuit complexity. On the other hand, there is a clear path for improvement in a smarter approach to traversing closed loops. Another issue is that the PAG for electronic circuits is large, complicated and difficult to understand. It is also computationally demanding to sample, since it requires many repeated sampling attempts. Nevertheless, the electronic circuits and the respective systems of ODEs that the PAG does manage to generate are guaranteed to be physically correct. Our attempt at automated modeling of electronic circuits demonstrates how to use the PAG framework to express highly complex types of background knowledge.

6.1.4 Bayesian updating

In Chapter 2, we introduced a simple Monte-Carlo algorithm for sampling PCFGs that enables the use of probabilistic grammars in equation discovery. In Chapter 3 and Chapter 4, we extended the framework of PCFGs with attributes to enable the encoding of more complex types of background knowledge, while still relying on randomly sampling expressions from grammars. In Chapter 5, we addressed the issue of algorithmic improvement by developing a Bayesian algorithm that iteratively updates the probabilities of production rule in a PCFG and guides the search towards more promising areas of the search space.

In the new algorithm, we begin with a PCFG with initial probabilities of production rules. Equation discovery proceeds in iterations, during which a number of random expression are sampled from the PCFG and evaluated. Based on the parse trees and errors-of-fit of the expressions we calculate new probabilities of production rules to be used in the next iteration.

We calculate new probabilities using the m-estimate, an estimate of the posterior probability that balances the evidence with the prior probability. By including only an expression with a low error in the calculation, we condition the posterior probability on expressions that fit the data well. In this way, the probability of successfully discovering the underlying equation increases with each iteration. The probability distribution defined by the PCFG with the optimized production probabilities can be interpreted as the posterior probability distribution over the space of expressions, combining prior beliefs (initial probabilities) and evidence (the data). An estimation of the posterior distribution improves the interpretability of equation discovery.

To test the Bayesian updating algorithm and analyze its behavior, we perform a small, illustrative computational experiment, involving the discovery of three hand-crafted equations that challenge the algorithm in various ways. The Bayesian algorithm outperforms Monte-Carlo sampling for all three equations by achieving a lower error with the same number of evaluated expressions. The production rule probabilities mostly evolve accord-

ing to our expectations, indicating that the proposed approach to updating probabilities indeed works.

On the other hand, due to its iterative nature, the Bayesian algorithm does not parallelize as well as Monte-Carlo sampling. As such, the Bayesian algorithm takes significantly longer to discover equations than random sampling when significant computational resources are available.

6.2 Discussion

The purpose of this work is to advance the field of equation discovery by enabling direct and intuitive control over the parsimony of generated mathematical expressions, introducing the various benefits of using a probabilistic framework and enabling the encoding of more complex types of background knowledge, which can significantly reduce the difficulty of discovering the governing equations for a given problem.

6.2.1 Parsimony and background knowledge

The PCFG framework for equation discovery handles parsimony elegantly and enables explicit control over it. As demonstrated by the theoretical analysis in Chapter 2, PCFGs prove themselves a powerful tool for expressing prior knowledge and impose both hard and soft constraints on the space of equations. However, the expressivity of PCFGs is limited by their lack of context awareness. This limitation is particularly evident when addressing systems of equations, such as in the domain of dynamical systems, since coupling between different equations necessitates some form of context awareness. For such problems, PCFGs as a framework for expressing background knowledge are inferior to frameworks such as process-based modelling.

The extension of PCFGs with attributes and attribute rules greatly increases the expressive power of the framework. PAGs enable the encoding of various types of background knowledge that are impossible to encode using PCFGs, as demonstrated by the examples in Chapter 4. Leveraging additional background knowledge, such as dimensional consistency, constrains the space of possible equations and improves the performance and computational efficiency of equation discovery.

6.2.2 Theoretical analysis and probability theory

An advantage of the relative simplicity of PCFGs lies in the potential for theoretical analysis and the direct application of probability theory, as demonstrated in the first sections of Chapter 2. Analyzing the properties of a PCFG enables us to predict the performance of equation discovery before committing to expensive computational experiments, therefore allowing us to tune the structure and/or parameters of the grammar to maximize the expected performance. Such analysis is much more difficult to perform for PAGs, since attribute rules introduce a lot of complexity to the sampling process and the theory of attribute grammars is a relatively unexplored topic. As such, analyzing the properties of PAGs for equation discovery currently requires empirical experiments, relying on the statistical analysis of a large sample of random expressions from a PAG.

6.2.3 Accessibility of PCFGs and PAGs

In realistic equation discovery use cases, frameworks for expressing prior knowledge are supposed to be used by domain experts and enable them to leverage their domain knowledge as efficiently as possible. To that end, an important quality of such frameworks is their

accessibility to experts from other domains. This represents an important weakness of the work, presented in this thesis.

To encode prior knowledge using the PCFG framework, a user has to write their own grammar, or modify an existing template. This is a nontrivial task to ask of users, since formal grammars are part of the curriculum only for computer scientists and some mathematicians. Granted, context-free grammars are built on rather simple concepts. Nevertheless, writing a well-functioning grammar for equation discovery requires some experience and practice, avoiding pitfalls such as endless recursion and grammars with incorrect mathematical syntax, as well as the ability to translate concepts from domain knowledge into production rules and symbols. This process includes both writing the structure of the grammar, as well as determining reasonable values of production rule probabilities. A possible solution lies in automated methods for learning PCFGs from collections of mathematical expressions [80].

These challenges are substantially bigger when writing a PAG. To fully leverage the expressive power of the PAG framework, a user must understand the basics of the sampling algorithm and the role of different types of attribute rules. In the current implementation of the framework, attribute rules are expressed as Python code. Therefore, the user must also know Python. On a conceptual level, the accessibility of the framework varies for different types of domain knowledge one wishes to express. For instance, measurement units and dimensional analysis are expressed very intuitively, since measurement units are encoded as attributes and are propagated using widely-understood mathematical rules. In contrast, the chemical kinetics example requires some attributes that do not directly represent concepts from domain knowledge, but are needed to control the flow of the generation algorithm. Writing a PAG for this example is conceptually not far from a software engineering task.

As such, it is not reasonable to expect domain experts to independently use the PAG framework. In its current state, a real-world use case for the PAG framework would require close collaboration between domain scientists and equation discovery experts (or at least computer scientists) in an interdisciplinary setting. On the other hand, we can view the presented framework as the first prototype of an expression-generating engine. Accessibility to experts from other domains can be improved by building interfaces that make use of the framework and provide a more intuitive front to users.

6.2.4 Computational efficiency

Many equation discovery approaches can discover the governing equations for a problem given infinite time and resources. For practical applicability of such systems, computational efficiency is critical. In this respect, the approaches presented in this thesis fall somewhat short. The Monte-Carlo sampling approach often requires testing a large number of candidate expressions, even when searching a constrained search space. This is not a serious issue for problems without numerical parameters. However, if a problem requires the estimation of parameter values, often computation on supercomputer clusters is required. For very complex problems, the probability of discovery in a reasonable time frame is infinitesimal.

Part of the reason for the lack of computational efficiency is unoptimized code. All algorithms and frameworks in this thesis are implemented in the publicly-accessible Python package ProGED. The program is written entirely in Python – a scripting language, which is famously slow for computationally demanding tasks. A ten-fold or even hundred-fold speedup could be achieved by optimizing the implementation in programming languages more suited to computation, such as C++ [81] or Rust [82], or by making use of tools such as Cython or Numba.

Another reason for long computation times is the simplicity of the naive Monte-Carlo algorithm. A better algorithm for searching the constrained space of equations is critical for improving the practical usefulness of the proposed approaches. The Bayesian updating algorithm, presented in Chapter 5, demonstrates a potential direction for further development. The algorithm clearly outperforms random sampling in most experiments. However, the Bayesian algorithm comes with a significant trade-off. Since the algorithm generates and tests candidate equations sequentially, it is much less open to parallel computation strategies. For instance, using the independent Monte-Carlo sampling, the parameter estimation of 100 to 1000 candidate equations can be performed in parallel on a supercomputing cluster. In this comparison, a sequential search algorithm must be 100 to 1000 times more efficient to find the solution faster. Even though the total processor time (and electric bill) is longer for Monte-Carlo sampling, the time-to-solution can be much lower than for smarter algorithms if powerful computational infrastructure is available.

6.3 Hypotheses

After summarizing and reviewing the contents of this thesis, we can now examine and answer the hypotheses set at the beginning of this work.

6.3.1 Hypothesis 1

We can design an equation discovery approach, based on probabilistic grammars (PCFGs and PAGs), that overcomes the limitations of existing approaches in ensuring parsimony, expressing different types of domain knowledge, and providing a probabilistic interpretation of results.

Confirmed. We designed two frameworks for equation discovery, one based on PCFGs and one based on PAGs. Both frameworks enable the encoding of various types of background knowledge and leverage it to constrain the space of possible equations. We demonstrated how probabilistic grammars inherently follow the parsimony principle and parametrize it intuitively through the probabilities of recursive production rules.

PCFGs are a powerful mathematical construct that already enables the encoding of many types of background knowledge. Their probabilistic interpretation and properties allow for fine control over the probability distribution, defined by the grammar, imposing soft constraints on the space of equations.

PAGs extend PCFGs with attributes and attributes rules, enhancing them with a limited form of context-sensitivity. This addresses the shortcomings of the expressive power of PCFGs, enabling users to express more complex types of background knowledge, such as measurement units and coupling terms in systems of equations.

Existing approaches to equation discovery typically enforce parsimony through regularization terms, which are unintuitive and difficult to interpret, or by imposing filtering and selection procedures on results, which can be difficult to justify in practical use cases. Many contemporary approaches are fully data-driven and do not attempt to leverage prior knowledge at all (deep learning). Those that do attempt to use prior knowledge, often do so only in a limited fashion, relying on simple libraries of terms and functions (sparse regression) or by setting weights for operators and variables (genetic programming, reinforcement learning). A number of approaches are based entirely on dimensional analysis, eschewing all other types of background knowledge. Finally, most existing approaches do not have a useful probabilistic interpretation. Existing Bayesian approaches provide robust estimates of posterior distributions, but fall short on the front of expressing background knowledge.

The approaches, introduced in this thesis, successfully address the parsimony principle, the expression of different types of background knowledge, and provide a robust probabilistic interpretation.

6.3.2 Hypothesis 2

We can implement the designed approach into a software tool, which enables the discovery of algebraic equations from data.

Confirmed. We successfully implemented all equation discovery approaches and algorithms introduced in this thesis in an open-source Python library called ProGED. The theoretical analysis and empirical experiments in Chapter 2 demonstrate that using PCFGs to encode background knowledge and define the search space of expressions, using a Monte-Carlo sampling algorithm to generate candidate expressions, and selecting the final equation based on error-of-fit, is an approach capable of discovering equations from data. The empirical experiments in Chapter 3 demonstrate that encoding dimensional-consistency in attributes, using an algorithm to transform the attribute grammar into a PCFG, and sampling it with the Monte-Carlo algorithm, enables the discovery of equations from data. In Chapter 4, we present PAGs for three different domains with examples of equations they generate. Coupled with the demonstrations of the methodology in Chapters 2 and 3, this shows that the direct sampling algorithm for PAGs is also capable of discovering equations from data. Finally, the experiment involving three target equations in Chapter 5 demonstrates the ability of the Bayesian grammar updating algorithm to discover equations.

6.3.3 Hypothesis 3

The developed approach can outperform existing equation discovery approaches in performance, computational efficiency and applicability.

Partially confirmed. Many existing approaches are applicable only to a certain class of equation discovery problems, i.e., problems, linear in parameters, algebraic equations, etc. In contrast, the probabilistic grammar-based approaches, introduced in this thesis, are widely applicable to all types of problems in equation discovery. They can discover equations of any mathematical form, nonlinear in parameters, and are applicable to algebraic equations, differential equations and systems of ODEs. The expressive power of PCFGs and especially PAGs empowers users to leverage various types of background knowledge to constrain the search space and thereby improve the performance and computational efficiency of equation discovery.

On the other hand, our computational experiments in Chapter 2 demonstrate that in the absence of useful background knowledge, the approach discovers fewer equations than competing algorithms, while requiring more computational power. Nevertheless, when useful background knowledge is leveraged, such as when encoding dimensional-consistency with PAGs, the approach performs competitively with state-of-the-art approaches in the field. In the course of this work, we have not performed the comparative computational experiments using the PAG framework from Chapter 4 necessary to test the algorithm's performance and computational efficiency in settings, where extensive domain knowledge can be leveraged (i.e., the dynamical systems and electronic circuits examples in Chapter 4).

To conclude, we can confirm that the introduced approaches outperform existing methods in applicability. However, the available evidence from computational experiments is insufficient to confirm or reject the hypothesis when it comes to equation discovery performance and computational efficiency.

6.4 Scientific Contributions

Overall, this PhD thesis advances the field of equation discovery by improving the methods for leveraging both general and domain-specific background knowledge.

1. **The initial contribution of this thesis is the innovative use of probabilistic context-free grammars in equation discovery. Employing PCFGs to represent background knowledge and generate candidate expressions yields simpler and more accurate equations and enables the use of soft constraints, fundamentally enhancing the equation discovery process.** The introduction of PCFGs brings several important improvements to equation discovery. Firstly, PCFGs exhibit inherent parsimony since the probability of an expression falls with expression complexity. This allows for the discovery of less complex equations, increasing the likelihood of scientifically relevant discoveries. Secondly, the probabilities of production rules enable a high degree of control over the derived expressions. This introduces soft constraints to equation discovery, which allow domain experts to not only specify which parts of the search space to include and which not to, but to finely control the probabilities of different parts of the search space. By increasing the flexibility and options available to experts when expressing domain knowledge, soft constraints will improve the performance of equation discovery. Lastly, a PCFG defines a probability distribution over the expressions it derives. This is a highly useful property that opens the door to other probabilistic approaches, such as those based on Bayesian statistics. The introduction of PCFGs for equation discovery can thus inspire novel probabilistic approaches that make use of PCFGs as the base of a probabilistic framework.

2. **The next contribution involves implementing probabilistic attribute grammars that enable dimensionally-consistent equation discovery. This approach maintains the expressive power of PCFGs, while eliminating physically meaningless equations, which significantly improves the performance and computational efficiency of the equation discovery process.** Dimensional analysis has a long tradition in physics, engineering and other domains and has been used in equation discovery before, both as the basis of ED algorithms, and as an upgrade to existing algorithms. However, combining dimensional consistency with other types of background knowledge has proven to be a challenge for background knowledge frameworks. In this thesis we introduce probabilistic attribute grammars that efficiently encode dimensional consistency through attributes and attribute rules, while leaving the structure and probabilities of the underlying PCFG free for expressing other background knowledge. Using extensive computational experiments we show that dimensional consistency significantly improves the performance of equation discovery on a database of equations from physics. Further improvements can be expected for use cases, where other types of background knowledge are encoded in the grammar in tandem with dimensional consistency.

3. **Next, the thesis introduces a general attribute grammar methodology coupled with a novel technique for directly sampling probabilistic attribute grammars. This novel approach can encode many complex types of background knowledge, extending the reach of domain knowledge in equation discovery.** PCFGs are a powerful framework for expressing background knowledge, but they have important limitations. For instance, it is impossible to guarantee the identity of two or more terms generated in an equation or system of equations without at least limited context information. We address this shortcoming by in-

troducing a general framework for using probabilistic attribute grammars to express background knowledge. We significantly improve the applicability of the approach, used in Contribution 2, which relies on transforming an attribute grammar into a context-free grammar, by developing an algorithm for the direct sampling of probabilistic attribute grammars. We demonstrate how to use the new framework in different domains using three concrete and detailed examples. The first of these is dimensional consistency, where we show how the new framework solves the problem from Contribution 2 in a more elegant way. The second example entails the generation of coupling terms in dynamical systems and more specifically, the complicated restrictions required in equations from chemical kinetics – both of which represent relevant and important problems. Finally, the third example is the identification of electronic circuits. Here, the example demonstrates the applicability of the new approach to domain knowledge that is notoriously complex and difficult to express.

4. **Finally, the development of an algorithm for the iterative Bayesian updating of grammar probabilities overcomes the limitations of simple random sampling and paves the way towards computationally efficient equation discovery. An additional benefit of the Bayesian approach is the approximation of the posterior distribution over mathematical expressions, which improves the interpretability of equation discovery.** Monte-Carlo sampling is the simplest method for generating candidate expressions from a probabilistic grammar, but it is computationally inefficient. To enable computationally efficient grammar-based equation discovery, more sophisticated algorithms are needed that require the evaluation of fewer candidate expressions. To that end, we develop a Bayesian algorithm that iteratively updates the probabilities of production rules in a PCFG using the m-estimate to guide the search towards more promising areas of the search space. We perform a small empirical experiment, demonstrating that the Bayesian algorithm outperforms Monte-Carlo sampling on average. Furthermore, we show that taking into account the results of previous iterations results in a grammar, whose distributions of production rules exhibit the properties of the target equation. Finally, we show that the Bayesian algorithm results in a posterior probability of the target equation, significantly greater than its prior probability, demonstrating that the method works as intended.

6.5 Further Work

The work presented in this thesis introduces novel approaches to equation discovery, with a focus on expressing and leveraging different types of background knowledge. We showed that the methods can successfully discover equations in different settings and can compete with existing methods when useful background knowledge is available. The open source implementation ProGED allows experts from other fields to use the developed methods to facilitate scientific discovery in their own work. Furthermore, the introduction of new paradigms opens a number of avenues for exciting further research.

To further investigate and validate the practical usefulness of the proposed approaches, a study of the effects of noise on the performance and efficiency of equation discovery is needed. Related work [83] has demonstrated the inherent limits of model selection in equation discovery in noisy settings. We expect the Monte-Carlo approach for generating candidate equations to be highly robust to noise and approach the theoretical limitations only in the model selection step. In contrast, Bayesian iterative updating can be much more vulnerable to disruptions due to local minima and noise. A detailed study

of these phenomena cannot only test and verify the usefulness of probabilistic grammar-based approaches, but potentially also give new insights into the behavior and limitations of equation discovery methods in noisy settings.

Testing and evaluating methods that leverage background knowledge is exceptionally difficult while relying on synthetic benchmark data, where background knowledge is absent or artificial. As such, real-world applicability can only be fairly evaluated through practical applications, preferably through interdisciplinary collaboration with domain experts. The performance and general usefulness of the proposed approaches depends on how much useful data is available, how much background knowledge is available, as well as how much of the background knowledge we can express and leverage using our framework. Therefore, practical applications of the probabilistic grammar-based approach to equation discovery are an important step in further research.

As evident from the computation experiments in Chapter 2 and Chapter 3, the main shortcoming of the approach is the inefficiency of the Monte-Carlo sampling algorithm. The work on a Bayesian updating algorithm in Chapter 5 shows a promising direction for algorithmic improvements. The empirical results show the first version of the m-estimate algorithm generally outperforms random sampling, but faces significant challenges. Firstly, the algorithm can get stuck in local minima, which can be alleviated by techniques for balancing exploration and exploitation. Secondly, a sequential updating algorithm is difficult to parallelize and therefore fully utilize available supercomputing resources. Techniques for sampling and evaluating equations in larger batches could help address this challenge. Finally, the algorithm relies on manipulating the probabilities of a grammar. In compact and elegantly-structured grammars, there are often only several such values to manipulate. It is an open question whether such low-dimensional parametrization is sufficient for the algorithm to fully express the bias learned by evaluating generated expressions and whether increasing the number of grammar parameters could improve algorithm performance. Further investigating and improving the grammar updating algorithm is therefore an important research direction to pursue. Besides further development to the proposed updating algorithm, alternatives can be considered, such as employing reinforcement learning to select production rules and guide the generation process.

The PAG framework, introduced in Chapter 4, is a promising and powerful framework for expressing background knowledge in equation discovery. Due to the relatively small existing body of research on attributed grammars, the details of its properties and behavior are not well understood. A theoretical consideration of the PAG framework and the properties of the probability distributions PAGs define would facilitate the adoption and usefulness of PAGs in equation discovery. Furthermore, the demonstrations of the framework in this work are only a few illustrative examples of its application. To better understand the power of PAGs for expressing background knowledge in equation discovery, a thorough examination of different types of background knowledge, and their encoding using PAGs, is required.

Finally, one of the barriers in the way of wider adoption of PCFGs and PAGs by domain experts is the difficulty of designing well-behaved grammars that express the desired aspects of background knowledge. To alleviate this, the development of higher-level, user-friendly interfaces should be considered. Alternatively, algorithms can be developed that automate the building of PCFGs or PAGs based on background knowledge encoded in existing systems for knowledge representation, such as knowledge graphs, ontologies or process-based models, which are more widely understood and adopted in the scientific community.

Appendix A

Feynman Database for Symbolic Regression

The Feynman database was constructed by Udrescu and Tegmark [13] to facilitate the development and testing of algorithms for symbolic regression. The database is composed of one hundred important equations from physics and acts as a good playground and benchmark dataset for equation discovery. The following table presents the one hundred equations from the Feynman database, along with their file name in the database, which is available at https://space.mit.edu/home/tegmark/aifeynman.html.

	Filename	Formula
0	I.6.2a	exp(-theta**2/2)/sqrt(2*pi)
1	I.6.2	exp(-(theta/sigma)**2/2)/(sqrt(2*pi)*sigma)
2	I.6.2b	exp(-((theta-theta1)/sigma)**2/2)/(sqrt(2*pi)*sigma)
3	I.8.14	sqrt((x2-x1)**2+(y2-y1)**2)
4	I.9.18	G*m1*m2/((x2-x1)**2+(y2-y1)**2+(z2-z1)**2)
5	I.10.7	m_0/sqrt(1-v**2/c**2)
6	I.11.19	x1*y1+x2*y2+x3*y3
7	I.12.1	mu*Nn
8	I.12.2	q1*q2*r/(4*pi*epsilon*r**3)
9	I.12.4	q1*r/(4*pi*epsilon*r**3)
10	I.12.5	q2*Ef
11	I.12.11	q*(Ef+B*v*sin(theta))
12	I.13.4	1/2*m*(v**2+u**2+w**2)
13	I.13.12	G*m1*m2*(1/r2-1/r1)
14	I.14.3	m*g*z
15	I.14.4	1/2*k_spring*x**2
16	I.15.3x	(x-u*t)/sqrt(1-u**2/c**2)
17	I.15.3t	(t-u*x/c**2)/sqrt(1-u**2/c**2)
18	I.15.1	m_0*v/sqrt(1-v**2/c**2)
19	I.16.6	(u+v)/(1+u*v/c**2)
20	I.18.4	(m1*r1+m2*r2)/(m1+m2)
21	I.18.12	r*F*sin(theta)
22	I.18.14	m*r*v*sin(theta)
23	I.24.6	1/2*m*(omega**2+omega_0**2)*1/2*x**2
24	I.25.13	q/C
25	I.26.2	arcsin(n*sin(theta2))
26	I.27.6	1/(1/d1+n/d2)
27	I.29.4	omega/c
28	I.29.16	sqrt(x1**2+x2**2-2*x1*x2*cos(theta1-theta2))

Continued on next page

	Filename	Formula
29	I.30.3	Int_0*sin(n*theta/2)**2/sin(theta/2)**2
30	I.30.5	arcsin(lambd/(n*d))
31	I.32.5	q**2*a**2/(6*pi*epsilon*c**3)
32	I.32.17	(1/2*epsilon*c*Ef**2)*(8*pi*r**2/3)*(omega**4/(omega**2-omega_0**2)**2)
33	I.34.8	q*v*B/p
34	I.34.1	omega_0/(1-v/c)
35	I.34.14	(1+v/c)/sqrt(1-v**2/c**2)*omega_0
36	I.34.27	(h/(2*pi))*omega
37	I.37.4	I1+I2+2*sqrt(I1*I2)*cos(delta)
38	I.38.12	4*pi*epsilon*(h/(2*pi))**2/(m*q**2)
39	I.39.1	3/2*pr*V
40	I.39.11	1/(gamm-1)*pr*V
41	I.39.22	n*kb*T/V
42	I.40.1	n_0*exp(-m*g*x/(kb*T))
43	I.41.16	h/(2*pi)*omega**3/(pi**2*c**2*(exp((h/(2*pi))*omega/(kb*T))-1))
44	I.43.16	mu_drift*q*Volt/d
45	I.43.31	mob*kb*T
46	I.43.43	1/(gamm-1)*kb*v/A
47	I.44.4	n*kb*T*ln(V2/V1)
48	I.47.23	sqrt(gamm*pr/rho)
49	I.48.2	m*c**2/sqrt(1-v**2/c**2)
50	I.50.26	x1*(cos(omega*t)+alpha*cos(omega*t)**2)
51	II.2.42	kappa*(T2-T1)*A/d
52	II.3.24	Pwr/(4*pi*r**2)
53	II.4.23	q/(4*pi*epsilon*r)
54	II.6.11	1/(4*pi*epsilon)*p_d*cos(theta)/r**2
55	II.6.15a	p_d/(4*pi*epsilon)*3*z/r**5*sqrt(x**2+y**2)
56	II.6.15b	p_d/(4*pi*epsilon)*3*cos(theta)*sin(theta)/r**3
57	II.8.7	3/5*q**2/(4*pi*epsilon*d)

Continued on next page

	Filename	Formula
58	II.8.31	epsilon*Ef**2/2
59	II.10.9	sigma_den/epsilon*1/(1+chi)
60	II.11.3	q*Ef/(m*(omega_0**2-omega**2))
61	II.11.17	n_0*(1+p_d*Ef*cos(theta)/(kb*T))
62	II.11.20	n_rho*p_d**2*Ef/(3*kb*T)
63	II.11.27	n*alpha/(1-(n*alpha/3))*epsilon*Ef
64	II.11.28	1+n*alpha/(1-(n*alpha/3))
65	II.13.17	1/(4*pi*epsilon*c**2)*2*I/r
66	II.13.23	rho_c_0/sqrt(1-v**2/c**2)
67	II.13.34	rho_c_0*v/sqrt(1-v**2/c**2)
68	II.15.4	-mom*B*cos(theta)
69	II.15.5	-p_d*Ef*cos(theta)
70	II.21.32	q/(4*pi*epsilon*r*(1-v/c))
71	II.24.17	sqrt(omega**2/c**2-pi**2/d**2)
72	II.27.16	epsilon*c*Ef**2
73	II.27.18	epsilon*Ef**2
74	II.34.2a	q*v/(2*pi*r)
75	II.34.2	q*v*r/2
76	II.34.11	g_*q*B/(2*m)
77	II.34.29a	q*h/(4*pi*m)
78	II.34.29b	g_*mom*B*Jz/(h/(2*pi))
79	II.35.18	n_0/(exp(mom*B/(kb*T))+exp(-mom*B/(kb*T)))
80	II.35.21	n_rho*mom*tanh(mom*B/(kb*T))
81	II.36.38	mom*H/(kb*T)+(mom*alpha)/(epsilon*c**2*kb*T)*M
82	II.37.1	mom*(1+chi)*B
83	II.38.3	Y*A*x/d
84	II.38.14	Y/(2*(1+sigma))
85	III.4.32	1/(exp((h/(2*pi))*omega/(kb*T))-1)
86	III.4.33	(h/(2*pi))*omega/(exp((h/(2*pi))*omega/(kb*T))-1)

Continued on next page

	Filename	Formula
87	III.7.38	2*mom*B/(h/(2*pi))
88	III.8.54	sin(E_n*t/(h/(2*pi)))**2
89	III.9.52	(p_d*Ef*t/(h/(2*pi)))*sin((omega-omega_0)*t/2)**2/((omega-omega_0)*t/2)**2
90	III.10.19	mom*sqrt(Bx**2+By**2+Bz**2)
91	III.12.43	n*(h/(2*pi))
92	III.13.18	2*E_n*d**2*k/(h/(2*pi))
93	III.14.14	I_0*(exp(q*Volt/(kb*T))-1)
94	III.15.12	2*U*(1-cos(k*d))
95	III.15.14	(h/(2*pi))**2/(2*E_n*d**2)
96	III.15.27	2*pi*alpha/(n*d)
97	III.17.37	bet*(1+alpha*cos(theta))
98	III.19.51	-m*q**4/(2*(4*pi*epsilon)**2*(h/(2*pi))**2)*(1/n**2)
99	III.21.20	-rho_c_0*q*A_vec/m

Appendix B

Detailed Experimental Results 1

The table provided in this section presents detailed results of the experiment described in Chapter 2. Each row corresponds to an equation discovery task from the Feynman database. For each problem, we performed six independent samplings of 10^5 candidate equations. Three of these were based on the uniform universal grammar (labelled U) and three were based on the biased universal grammar (labelled B). The columns of the table, from left to right, are as follows;

- Index of the problem from the Feynman dataset.
- #v. Number of variables in the target expression.
- #p. Number of constant parameters in the target expression.
- #o. Number of mathematical operations or special function in the target expression.
- #c. Number of characters in the string representation of the target expression.
- p_U. The probability of generating the target expression, using the uniform universal grammar. Approximated by the probability of a single parse tree. In other words, the approximation ignores the semantic ambiguity of the grammar.
- p_B. Same as p_U, but using the biased universal grammar.
- $\#S_U$. Number of successes in three independent samplings, using the uniform universal grammar. A sampling is successful if it finds at least one model with $RRMSE < 10^{-9}$.
- $\#S_B$. Same as $\#S_U$, but using the biased universal grammar.
- $\#N_U$. Number of unique expressions in the canonical form, generated using the uniform universal grammar. Formatted as a triplet of values, each corresponding to one of three independent samplings. Expressed in the units of thousands.
- $\#N_B$. Same as $\#N_U$, but using the biased universal grammar.
- cov_U. Sum of probabilities (coverage) of all unique expressions, sampling using the uniform universal grammar. Formatted as a triplet of values, each corresponding to one of three independent samplings.
- cov_B. Same as cov_U, but using the biased universal grammar.

	#v	#p	#o	#c	p_U	p_B	$\#S_U$	$\#S_B$	$N_U[10^3]$	$N_B[10^3]$	cov_U	cov_B
0	1	2	6	27	$1.1{\cdot}10^{-6}$	$2.1{\cdot}10^{-7}$	3	2	(30, 30, 30)	(14, 24, 3)	(0.38, 0.38, 0.38)	(0.21, 0.54, 0.29)
1	2	2	8	43	$1.7{\cdot}10^{-11}$	$7{\cdot}10^{-12}$	0	0	(35, 35, 25)	(30, 30, 20)	(0.38, 0.38, 0.31)	(0.54, 0.54, 0.34)
2	3	2	9	52	$1.2{\cdot}10^{-19}$	$1.5{\cdot}10^{-18}$	0	0	(38, 38, 38)	(34, 25, 35)	(0.37, 0.37, 0.37)	(0.52, 0.47, 0.52)
3	4	0	4	27	$2.2{\cdot}10^{-24}$	$7.7{\cdot}10^{-21}$	0	0	(40, 41, 30)	(38, 37, 28)	(0.36, 0.36, 0.3)	(0.51, 0.51, 0.45)
4	9	0	8	42	$6.6{\cdot}10^{-46}$	$2.5{\cdot}10^{-39}$	0	0	(46, 46, 46)	(46, 46, 46)	(0.33, 0.33, 0.33)	(0.47, 0.47, 0.47)
5	3	1	4	21	$1.7{\cdot}10^{-11}$	$3.8{\cdot}10^{-11}$	0	0	(38, 38, 38)	(34, 34, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
6	6	0	5	17	$3.6{\cdot}10^{-12}$	$3.8{\cdot}10^{-11}$	0	0	(44, 44, 44)	(42, 42, 42)	(0.35, 0.35, 0.35)	(0.49, 0.49, 0.49)
7	2	0	1	5	$2.9{\cdot}10^{-3}$	$8.8{\cdot}10^{-3}$	3	3	(35, 35, 35)	(30, 31, 30)	(0.37, 0.37, 0.37)	(0.54, 0.54, 0.53)
8	4	1	6	27	$2.7{\cdot}10^{-15}$	$1.7{\cdot}10^{-13}$	0	1	(40, 40, 41)	(37, 38, 28)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.43)
9	3	1	5	24	$7.5{\cdot}10^{-13}$	$2.5{\cdot}10^{-11}$	1	3	(38, 38, 38)	(34, 35, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
10	2	0	1	5	$2.9{\cdot}10^{-3}$	$8.8{\cdot}10^{-3}$	3	3	(35, 35, 30)	(30, 30, 10)	(0.38, 0.37, 0.36)	(0.53, 0.53, 0.07)
11	5	0	5	21	$2.3{\cdot}10^{-13}$	$4{\cdot}10^{-12}$	0	0	(42, 42, 30)	(40, 30, 30)	(0.35, 0.35, 0.24)	(0.5, 0.38, 0.39)
12	4	1	5	22	$9.6{\cdot}10^{-16}$	$2.3{\cdot}10^{-14}$	0	0	(40, 40, 40)	(38, 38, 27)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.31)
13	5	2	6	19	10^{-13}	$6.8{\cdot}10^{-13}$	0	0	(43, 42, 32)	(40, 40, 40)	(0.35, 0.35, 0.26)	(0.5, 0.5, 0.5)
14	3	0	2	5	$3.4{\cdot}10^{-5}$	$2{\cdot}10^{-4}$	3	3	(38, 38, 28)	(34, 25, 34)	(0.37, 0.37, 0.27)	(0.52, 0.45, 0.52)
15	2	1	3	17	$1.3{\cdot}10^{-7}$	$9.4{\cdot}10^{-7}$	3	3	(35, 35, 25)	(30, 31, 30)	(0.37, 0.37, 0.25)	(0.54, 0.54, 0.53)
16	4	1	6	25	$1.4{\cdot}10^{-17}$	$3.3{\cdot}10^{-16}$	0	0	(40, 41, 31)	(38, 38, 37)	(0.36, 0.36, 0.23)	(0.51, 0.51, 0.51)
17	4	1	7	30	$5.5{\cdot}10^{-21}$	$2.2{\cdot}10^{-19}$	0	0	(41, 41, 31)	(38, 27, 18)	(0.36, 0.36, 0.3)	(0.51, 0.33, 0.26)
18	3	1	5	23	$4.5{\cdot}10^{-13}$	$1.9{\cdot}10^{-12}$	0	0	(38, 38, 8)	(35, 35, 24)	(0.37, 0.37, 0.03)	(0.52, 0.52, 0.37)
19	3	1	5	18	$3.9{\cdot}10^{-15}$	$3.6{\cdot}10^{-14}$	0	0	(38, 28, 30)	(35, 34, 24)	(0.37, 0.29, 0.26)	(0.52, 0.52, 0.38)
20	4	0	5	21	$8.6{\cdot}10^{-15}$	$3.7{\cdot}10^{-13}$	0	0	(40, 41, 20)	(37, 37, 10)	(0.36, 0.36, 0.18)	(0.51, 0.51, 0.13)
21	3	0	3	14	$4.9{\cdot}10^{-7}$	$1.8{\cdot}10^{-6}$	3	3	(38, 38, 28)	(34, 35, 34)	(0.37, 0.37, 0.25)	(0.52, 0.52, 0.52)
22	4	0	4	16	$4.1{\cdot}10^{-9}$	$2.9{\cdot}10^{-8}$	1	2	(40, 41, 20)	(38, 37, 10)	(0.36, 0.36, 0.22)	(0.51, 0.51, 0.21)
23	4	1	7	36	$1.6{\cdot}10^{-15}$	$7.7{\cdot}10^{-14}$	0	0	(41, 41, 31)	(38, 37, 28)	(0.36, 0.36, 0.18)	(0.51, 0.51, 0.37)
24	2	1	1	3	$2.9{\cdot}10^{-3}$	$5.9{\cdot}10^{-3}$	3	3	(35, 25, 30)	(30, 31, 30)	(0.38, 0.32, 0.36)	(0.54, 0.54, 0.54)
25	2	0	4	21	0	0	0	0	(35, 35, 35)	(30, 21, 30)	(0.38, 0.37, 0.37)	(0.53, 0.19, 0.53)
26	3	2	4	13	$1.9{\cdot}10^{-9}$	$9.5{\cdot}10^{-10}$	1	0	(38, 38, 38)	(34, 34, 24)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.36)
27	2	0	1	7	$2.9{\cdot}10^{-3}$	$5.9{\cdot}10^{-3}$	3	3	(35, 35, 25)	(30, 30, 30)	(0.37, 0.37, 0.33)	(0.53, 0.53, 0.53)
28	4	1	8	44	$1.7{\cdot}10^{-19}$	$3.2{\cdot}10^{-18}$	0	0	(40, 31, 40)	(38, 38, 28)	(0.36, 0.24, 0.36)	(0.51, 0.51, 0.39)

Continued on next page

	#v	#p	#o	#c	p_U	p_B	$\#S_U$	$\#S_B$	$N_U[10^3]$	$N_B[10^3]$	cov_U	cov_B
29	3	4	7	39	$3{\cdot}10^{-23}$	$1.7{\cdot}10^{-24}$	0	0	(38, 38, 28)	(34, 34, 24)	(0.37, 0.37, 0.3)	(0.52, 0.52, 0.46)
30	3	0	4	19	0	0	0	0	(38, 38, 20)	(35, 35, 35)	(0.37, 0.37, 0.22)	(0.52, 0.52, 0.52)
31	4	1	5	29	$5.3{\cdot}10^{-17}$	$6.5{\cdot}10^{-15}$	0	0	(41, 41, 20)	(38, 37, 38)	(0.36, 0.36, 0.22)	(0.51, 0.51, 0.51)
32	6	1	11	71	$1.9{\cdot}10^{-39}$	$3.6{\cdot}10^{-34}$	0	0	(43, 44, 44)	(42, 42, 32)	(0.35, 0.35, 0.35)	(0.49, 0.49, 0.35)
33	4	0	3	7	$2.9{\cdot}10^{-7}$	$2.2{\cdot}10^{-6}$	3	3	(41, 40, 21)	(37, 30, 37)	(0.36, 0.36, 0.22)	(0.51, 0.39, 0.51)
34	3	1	3	15	$2.4{\cdot}10^{-8}$	$8.7{\cdot}10^{-8}$	0	0	(38, 38, 28)	(35, 24, 35)	(0.37, 0.37, 0.15)	(0.52, 0.46, 0.52)
35	3	2	7	33	$8.2{\cdot}10^{-18}$	$2.7{\cdot}10^{-17}$	0	0	(38, 39, 10)	(34, 34, 35)	(0.37, 0.37, 0.09)	(0.52, 0.52, 0.52)
36	2	1	3	16	$2.3{\cdot}10^{-4}$	$2.3{\cdot}10^{-4}$	3	3	(35, 35, 25)	(30, 30, 30)	(0.38, 0.37, 0.23)	(0.54, 0.54, 0.53)
37	3	1	7	30	$1.4{\cdot}10^{-13}$	$1.5{\cdot}10^{-13}$	0	0	(38, 38, 30)	(34, 35, 20)	(0.37, 0.37, 0.26)	(0.52, 0.52, 0.23)
38	4	1	7	35	$9.2{\cdot}10^{-12}$	$5.5{\cdot}10^{-11}$	0	0	(40, 40, 30)	(38, 38, 28)	(0.36, 0.36, 0.25)	(0.51, 0.51, 0.39)
39	2	1	3	8	$2.3{\cdot}10^{-4}$	$3.4{\cdot}10^{-4}$	3	3	(35, 35, 35)	(30, 30, 10)	(0.38, 0.38, 0.38)	(0.54, 0.53, 0.1)
40	3	2	4	15	$1.9{\cdot}10^{-9}$	$5{\cdot}10^{-9}$	2	1	(38, 38, 38)	(34, 34, 24)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.27)
41	4	0	3	8	$2.9{\cdot}10^{-7}$	$2.2{\cdot}10^{-6}$	3	3	(40, 40, 41)	(38, 38, 27)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.46)
42	6	1	7	22	$1.2{\cdot}10^{-14}$	$6.5{\cdot}10^{-14}$	0	0	(43, 43, 44)	(42, 42, 32)	(0.35, 0.35, 0.35)	(0.49, 0.49, 0.36)
43	5	3	13	63	$1.3{\cdot}10^{-25}$	$1.3{\cdot}10^{-24}$	0	0	(42, 42, 32)	(40, 30, 30)	(0.35, 0.35, 0.26)	(0.5, 0.29, 0.33)
44	4	0	3	17	$2.9{\cdot}10^{-7}$	$2.2{\cdot}10^{-6}$	3	3	(41, 41, 41)	(37, 38, 37)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
45	3	0	2	8	$3.4{\cdot}10^{-5}$	$2{\cdot}10^{-4}$	3	3	(38, 28, 38)	(34, 34, 35)	(0.37, 0.27, 0.37)	(0.52, 0.52, 0.52)
46	4	2	5	17	$1.6{\cdot}10^{-11}$	$5.4{\cdot}10^{-11}$	0	0	(41, 40, 20)	(38, 27, 38)	(0.36, 0.36, 0.15)	(0.51, 0.35, 0.51)
47	5	0	4	16	0	0	0	0	(42, 42, 32)	(40, 40, 40)	(0.35, 0.35, 0.27)	(0.5, 0.5, 0.5)
48	3	0	3	17	$4.9{\cdot}10^{-7}$	$1.2{\cdot}10^{-6}$	3	2	(38, 38, 20)	(35, 34, 35)	(0.37, 0.37, 0.18)	(0.52, 0.52, 0.52)
49	3	1	5	24	$1.2{\cdot}10^{-14}$	$9.9{\cdot}10^{-14}$	0	0	(38, 38, 38)	(35, 35, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
50	4	0	7	39	$8.3{\cdot}10^{-20}$	$6.2{\cdot}10^{-18}$	0	0	(41, 31, 41)	(37, 38, 38)	(0.36, 0.31, 0.36)	(0.51, 0.51, 0.51)
51	5	0	4	17	$1.6{\cdot}10^{-11}$	$6.9{\cdot}10^{-10}$	0	0	(42, 42, 42)	(40, 30, 40)	(0.35, 0.35, 0.35)	(0.5, 0.4, 0.5)
52	2	1	3	15	$9.2{\cdot}10^{-6}$	$7.7{\cdot}10^{-6}$	3	3	(35, 35, 35)	(30, 30, 30)	(0.37, 0.37, 0.38)	(0.53, 0.53, 0.53)
53	3	1	4	18	$3.9{\cdot}10^{-8}$	$1.9{\cdot}10^{-7}$	3	3	(38, 38, 38)	(35, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
54	4	1	7	36	$6.6{\cdot}10^{-12}$	$1.3{\cdot}10^{-11}$	0	0	(41, 41, 40)	(37, 38, 37)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
55	6	1	9	43	$3.9{\cdot}10^{-26}$	$1.1{\cdot}10^{-24}$	0	0	(43, 43, 40)	(42, 42, 42)	(0.35, 0.35, 0.33)	(0.49, 0.49, 0.49)
56	4	1	9	47	$3.8{\cdot}10^{-17}$	$1.1{\cdot}10^{-16}$	0	0	(40, 30, 40)	(38, 37, 38)	(0.36, 0.3, 0.36)	(0.51, 0.51, 0.51)
57	3	1	6	25	$7.3{\cdot}10^{-8}$	$1.8{\cdot}10^{-7}$	3	3	(38, 30, 38)	(34, 34, 35)	(0.37, 0.28, 0.37)	(0.52, 0.52, 0.52)

Continued on next page

	#v	#p	#o	#c	p_U	p_B	$\#S_U$	$\#S_B$	$N_U[10^3]$	$N_B[10^3]$	cov_U	cov_B
58	2	1	2	15	$9.2{\cdot}10^{-6}$	$1.7{\cdot}10^{-5}$	3	3	(35, 35, 35)	(31, 30, 30)	(0.37, 0.37, 0.38)	(0.54, 0.54, 0.54)
59	3	1	4	27	$2.4{\cdot}10^{-8}$	$3.7{\cdot}10^{-8}$	1	2	(38, 38, 38)	(35, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
60	5	0	4	30	$4.2{\cdot}10^{-15}$	$4.3{\cdot}10^{-13}$	0	0	(42, 42, 42)	(40, 40, 40)	(0.35, 0.35, 0.35)	(0.5, 0.5, 0.5)
61	6	1	7	32	10^{-16}	$7{\cdot}10^{-16}$	0	0	(44, 44, 44)	(42, 42, 42)	(0.35, 0.35, 0.35)	(0.49, 0.49, 0.49)
62	5	1	5	24	$2.4{\cdot}10^{-12}$	$1.4{\cdot}10^{-11}$	0	0	(42, 42, 32)	(40, 40, 40)	(0.35, 0.35, 0.23)	(0.5, 0.5, 0.5)
63	4	2	7	34	$6.4{\cdot}10^{-15}$	$7.9{\cdot}10^{-14}$	0	0	(41, 41, 40)	(37, 37, 38)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
64	2	3	6	25	$1.2{\cdot}10^{-11}$	$9.9{\cdot}10^{-12}$	0	0	(35, 35, 35)	(30, 30, 30)	(0.37, 0.38, 0.38)	(0.53, 0.53, 0.53)
65	4	1	7	27	$6.6{\cdot}10^{-12}$	$7.7{\cdot}10^{-11}$	0	0	(38, 38, 38)	(36, 36, 36)	(0.26, 0.26, 0.26)	(0.35, 0.35, 0.35)
66	3	1	4	25	$1.7{\cdot}10^{-11}$	$3.8{\cdot}10^{-11}$	0	0	(38, 38, 38)	(35, 34, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
67	3	1	5	27	$4.5{\cdot}10^{-13}$	$1.9{\cdot}10^{-12}$	0	0	(38, 38, 38)	(34, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
68	3	1	4	17	$3.9{\cdot}10^{-8}$	$7{\cdot}10^{-8}$	3	2	(38, 38, 38)	(35, 34, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
69	3	1	4	18	$3.9{\cdot}10^{-8}$	$7{\cdot}10^{-8}$	3	2	(38, 38, 38)	(34, 34, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
70	5	2	7	26	$1.5{\cdot}10^{-15}$	$2.4{\cdot}10^{-14}$	0	0	(43, 42, 42)	(40, 40, 40)	(0.35, 0.35, 0.35)	(0.5, 0.5, 0.5)
71	3	1	4	30	$4.5{\cdot}10^{-13}$	$1.3{\cdot}10^{-12}$	0	0	(38, 38, 38)	(34, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
72	3	0	2	15	$9.1{\cdot}10^{-7}$	10^{-5}	3	3	(38, 38, 30)	(35, 35, 25)	(0.37, 0.37, 0.22)	(0.52, 0.52, 0.27)
73	2	0	1	13	$1.2{\cdot}10^{-4}$	$6.8{\cdot}10^{-4}$	3	3	(35, 35, 35)	(30, 30, 30)	(0.38, 0.37, 0.38)	(0.54, 0.53, 0.54)
74	3	1	4	12	$2.7{\cdot}10^{-6}$	$5.2{\cdot}10^{-6}$	3	3	(38, 38, 38)	(34, 34, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
75	3	1	3	7	$2.7{\cdot}10^{-6}$	$5.2{\cdot}10^{-6}$	3	3	(38, 38, 38)	(34, 24, 35)	(0.37, 0.37, 0.37)	(0.52, 0.28, 0.52)
76	4	1	4	12	$2.3{\cdot}10^{-8}$	$5.6{\cdot}10^{-8}$	3	3	(40, 41, 40)	(38, 28, 28)	(0.36, 0.36, 0.36)	(0.51, 0.26, 0.4)
77	3	1	4	12	$2.7{\cdot}10^{-6}$	$3.4{\cdot}10^{-6}$	3	3	(38, 38, 38)	(35, 35, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
78	5	1	6	22	$1.5{\cdot}10^{-10}$	$1.1{\cdot}10^{-9}$	0	1	(42, 42, 42)	(40, 40, 40)	(0.35, 0.35, 0.35)	(0.5, 0.5, 0.5)
79	5	1	11	42	$1.8{\cdot}10^{-23}$	$1.6{\cdot}10^{-22}$	0	0	(42, 42, 42)	(40, 40, 40)	(0.35, 0.35, 0.35)	(0.5, 0.5, 0.5)
80	5	0	5	28	0	0	0	0	(42, 42, 42)	(40, 40, 40)	(0.35, 0.35, 0.35)	(0.5, 0.5, 0.5)
81	8	0	10	46	$1.6{\cdot}10^{-26}$	$1.2{\cdot}10^{-23}$	0	0	(45, 46, 46)	(44, 45, 45)	(0.34, 0.34, 0.34)	(0.47, 0.47, 0.47)
82	3	1	3	13	$2.4{\cdot}10^{-8}$	$8.3{\cdot}10^{-8}$	3	3	(38, 38, 38)	(34, 25, 35)	(0.37, 0.37, 0.37)	(0.52, 0.37, 0.52)
83	4	0	3	7	$2.9{\cdot}10^{-7}$	$2.2{\cdot}10^{-6}$	2	3	(41, 40, 41)	(38, 38, 38)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
84	2	2	3	15	$1.6{\cdot}10^{-7}$	$6.3{\cdot}10^{-8}$	3	3	(35, 35, 20)	(30, 30, 30)	(0.37, 0.37, 0.14)	(0.54, 0.54, 0.53)
85	4	3	8	34	$1.8{\cdot}10^{-14}$	$8.3{\cdot}10^{-15}$	0	0	(41, 41, 41)	(38, 38, 38)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
86	4	3	11	49	$7.3{\cdot}10^{-18}$	$8.1{\cdot}10^{-18}$	0	0	(40, 40, 40)	(38, 38, 38)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)

Continued on next page

	#v	#p	#o	#c	p_U	p_B	$\#S_U$	$\#S_B$	$N_U[10^3]$	$N_B[10^3]$	cov_U	cov_B
87	3	1	5	18	$2.7{\cdot}10^{-6}$	$5.2{\cdot}10^{-6}$	3	3	(38, 38, 38)	(34, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
88	3	2	5	24	$8.6{\cdot}10^{-16}$	$1.4{\cdot}10^{-15}$	0	0	(38, 38, 38)	(35, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
89	6	5	14	74	$1.4{\cdot}10^{-51}$	$1.1{\cdot}10^{-47}$	0	0	(44, 43, 43)	(42, 42, 42)	(0.35, 0.35, 0.35)	(0.49, 0.49, 0.49)
90	4	0	4	27	$1.2{\cdot}10^{-14}$	$1.5{\cdot}10^{-13}$	0	0	(40, 40, 40)	(38, 37, 38)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
91	2	1	3	12	$2.3{\cdot}10^{-4}$	$2.3{\cdot}10^{-4}$	3	3	(35, 35, 35)	(31, 30, 30)	(0.37, 0.37, 0.37)	(0.53, 0.53, 0.53)
92	4	1	6	23	$4.6{\cdot}10^{-10}$	$3.2{\cdot}10^{-9}$	0	0	(40, 41, 41)	(38, 38, 38)	(0.36, 0.36, 0.36)	(0.51, 0.51, 0.51)
93	5	1	6	26	$1.9{\cdot}10^{-14}$	$1.6{\cdot}10^{-13}$	0	0	(42, 42, 32)	(39, 40, 40)	(0.35, 0.35, 0.22)	(0.5, 0.5, 0.5)
94	3	2	5	16	$2.7{\cdot}10^{-11}$	$6.7{\cdot}10^{-11}$	0	0	(38, 38, 38)	(35, 25, 35)	(0.37, 0.37, 0.37)	(0.52, 0.26, 0.52)
95	3	1	5	26	$1.9{\cdot}10^{-9}$	$6{\cdot}10^{-9}$	2	2	(38, 38, 38)	(35, 35, 34)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
96	3	1	4	16	$2.7{\cdot}10^{-6}$	$3.4{\cdot}10^{-6}$	3	3	(38, 38, 38)	(35, 34, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
97	3	1	4	24	$3.4{\cdot}10^{-10}$	$7.5{\cdot}10^{-10}$	2	0	(38, 38, 38)	(34, 34, 35)	(0.37, 0.37, 0.37)	(0.52, 0.52, 0.52)
98	5	1	11	52	$2.5{\cdot}10^{-21}$	$1.7{\cdot}10^{-19}$	0	0	(42, 42, 42)	(40, 40, 40)	(0.35, 0.35, 0.35)	(0.5, 0.5, 0.5)
99	4	1	4	18	$2.3{\cdot}10^{-8}$	$8.3{\cdot}10^{-8}$	1	3	(30, 41, 41)	(37, 37, 37)	(0.32, 0.36, 0.36)	(0.51, 0.51, 0.51)

Appendix C

Detailed Experimental Results 2

In Chapter 3, we report on computational experiments that compare the performance of ProGED using a universal arithmetic PCFG and its dimensionally-consistent version, as well as DSO, a state-of-the-art deep learning method. The following table gives the expression with the lowest error, generated by the dimensionally-consistent grammar, for each problem from the Feynman database, as well as its ReRMSE. We rounded the values of the error and the numerical parameters to three decimals for easier reading.

	filename	error	expression
0	I.6.2a	6.17E-15	0.399*exp(-0.5*theta**2)
1	I.6.2	4.95E-03	-0.171/theta - 1.058/(sigma*(0.133 - exp(theta/sigma)))
2	I.6.2b	2.61E-02	0.416 - 0.291*(sigma + sin(0.282*sigma/theta1 + theta + theta1**2 - 7.2*theta1 + 161.576 - 8.2/theta1)/sigma)/sigma
3	I.8.14	6.25E-01	-2.164*x1*(tanh(0.286*exp(1.784*x2/x1)) - 1.094 - 0.022*y1/x2) - x1 + x2
4	I.9.18	6.49E-02	1.026*G*m1*m2/(x1*z1)
5	I.10.7	0.00E+00	m_0/(1 - v**2/c**2)**0.5
6	I.11.19	2.21E+00	x1*y1 + 1.008*x2*y3 + 0.963*x3*y2
7	I.12.1	0.00E+00	Nn*mu
8	I.12.2	3.50E-17	0.08*q1*q2/(epsilon*r**2)
9	I.12.4	1.08E-17	0.08*q1/(epsilon*r**2)
10	I.12.5	9.63E-12	Ef*q2
11	I.12.11	9.11E-16	q*(B*v*sin(theta) + Ef)
12	I.13.4	4.56E+00	0.344*m*(2*u + 1.091*w)*(v + 0.548*w)
13	I.13.12	6.78E+00	-0.842*G*m1*m2/(r1 - 5.406*r2)
14	I.14.3	0.00E+00	1.0*g*m*z
15	I.14.4	0.00E+00	0.5*k_spring*x**2
16	I.15.3x	4.80E-03	1.001*(-t*u + x)/cos(u/c)
17	I.15.3t	1.96E-03	0.998*t + u*(t - 1.844*x/u)/(c*(2.037*c/u - 0.907))
18	I.15.1	7.88E-13	m_0*v/cos(asin(v/c) - 25.133)
19	I.16.6	7.46E-02	c + 0.672*(1.018*c - v)*cos(0.556*c/u + 13.602)
20	I.18.4	2.22E-08	r2 + (r1 - 1.0*r2)/(1.0 + 1.0*m2/m1)
21	I.18.12	2.32E-16	1.0*F*r*sin(theta)
22	I.18.14	9.11E-16	m*r*v*sin(1.0*theta)
23	I.24.6	3.77E+00	0.532*m*omega*omega_0*x**2
24	I.25.13	0.00E+00	1.0*q/cp
25	I.26.2	6.82E-03	1.064*tan(0.908*n*sin(theta2))

	filename	error	expression
26	I.27.6	7.98E-17	d2/(n + 1.0*d2/d1)
27	I.29.4	0.00E+00	1.0*omega/c
28	I.29.16	9.29E-01	x1*cos(0.703*theta1 - 0.734*theta2 + 3.232) + 0.739*x1 + x2
29	I.30.3	1.12E+00	-0.802*Int_0*cos(n*theta) + 0.597*Int_0 + 0.597*Int_0/theta
30	I.30.5	6.19E-04	sinh(0.123 + 0.971*lambd/(d*n)) - 0.122
31	I.32.5	none	
32	I.32.17	3.77E+00	0.198*Ef**2*c*epsilon*r**2
33	I.34.8	1.52E-15	B*q*v/p
34	I.34.1	0.00E+00	omega_0/(1 - v/c)
35	I.34.14	4.48E-15	omega_0*(1.0 + v/c)/(1 - v**2/c**2)**0.5
36	I.34.27	1.29E-16	0.5*h*omega/pi
37	I.37.4	2.64E-01	-(1.756*I1 + 1.737*I2)*sin(0.027*delta**3 - delta + sin(0.189*delta) - 13.931)
38	I.38.12	6.28E-15	0.318*epsilon*h**2/(m*q**2)
39	I.39.1	1.07E-15	V*(0.416*pr + pr*sinh(1)**0.5)
40	I.39.11	4.90E-16	V*pr/(gamm - 1)
41	I.39.22	1.03E-15	T*kb*n/V
42	I.40.1	4.73E-01	-0.012*n_0*(5.783*g - 28.716) + 0.052
43	I.41.16	2.73E+01	T*kb*omega**2/c**2
44	I.43.16	1.39E-15	Volt*mu_drift*q/d
45	I.43.31	2.66E-15	1.0*T*kb*mob
46	I.43.43	2.75E-16	1.0*kb*v/(A*(gamm - 1))
47	I.44.4	1.71E+00	-8.269*T*kb*n/(7.82*V2/(0.042*V1*n + 0.835*V1 - V2) + n)
48	I.47.23	none	
49	I.48.2	2.76E-03	c**2*m/(atan(1.662*cos(v/c) - 0.84) + 0.312)

	filename	error	expression
50	I.50.26	2.06E+00	1.435*x1/(-alpha + 4.227 + 2.905*cos(omega*t + 28.269)/alpha)
51	II.2.42	7.99E-16	1.0*A*kappa*(-T1 + T2)/d
52	II.3.24	2.30E-17	0.08*Pwr/r**2
53	II.4.23	1.79E-17	0.08*q/(epsilon*r)
54	II.6.11	6.71E-16	0.08*p_d*cos(theta)/(epsilon*r**2)
55	II.6.15a	2.87E-01	p_d/(epsilon*r**3)
56	II.6.15b	2.71E-02	-0.108*p_d*(theta - 1.62)/(epsilon*r**3)
57	II.8.7	1.44E-17	0.048*q**2/(d*epsilon)
58	II.8.31	0.00E+00	0.5*Ef**2*epsilon
59	II.10.9	3.49E-17	sigma_den/(epsilon*(chi + 1.0))
60	II.11.3	2.46E-02	1.239*Ef*q/(m*omega_0**2)
61	II.11.17	1.82E+00	0.318*n_0*(-theta - 0.005*tan(16.492*n_0) + 6.942/theta)
62	II.11.20	none	
63	II.11.27	2.85E-14	Ef*alpha*epsilon*n/(-0.333*alpha*n + 1)
64	II.11.28	1.94E-08	-3.0*n/(n - 3.0/alpha) + 1.0
65	II.13.17	8.53E-03	0.464/(c**2*epsilon*r)
66	II.13.23	0.00E+00	rho_c_0/(1 - v**2/c**2)**0.5
67	II.13.34	5.38E-06	1.0*rho_c_0*v/sin(asin(1.0*v/c) - 10.996)
68	II.15.4	3.10E-16	-B*mom*cos(theta)
69	II.15.5	3.51E-16	-Ef*p_d*cos(theta)
70	II.21.32	9.39E-17	q/(epsilon*r*(12.566 - 12.566*v/c))
71	II.24.17	1.17E-04	omega*cos(1.28*tan(2.434*c/(d*omega)) + 0.002)/c
72	II.27.16	1.05E-14	1.0*Ef**2*c*epsilon
73	II.27.18	0.00E+00	1.0*Ef**2*epsilon
74	II.34.2a	3.12E-16	q*v/(r*cos(1) + 5.743*r)
75	II.34.2	1.43E-15	0.5*q*r*v
76	II.34.11	6.13E-16	0.5*B*g_*q/m

	filename	error	expression
77	II.34.29a	1.29E-16	0.08*h*q/m
78	II.34.29b	1.06E-13	6.283*B*Jz*g_*mom/h
79	II.35.18	3.85E-01	-0.217*mom - 0.124*n_0**2 + 0.786*n_0
80	II.35.21	1.68E+00	0.73*mom*n_rho*atan(0.585*B + 0.212)
81	II.36.38	1.72E+00	alpha*(-0.463*T - 0.837) + 2*alpha + 1.217
82	II.37.1	3.38E-15	B*(mom*(chi - 1.0) + 2*mom)
83	II.38.3	0.00E+00	A*Y*x/d
84	II.38.14	0.00E+00	Y/(2*sigma + 2.0)
85	III.4.32	none	
86	III.4.33	2.94E-03	-0.944*T*kb - 1.944*kb*(-T + T/(24.065*T*kb/(h*omega) + 0.834))
87	III.7.38	1.18E-14	-4.498*B*mom/(-h + h/tan(1))
88	III.8.54	none	
89	III.9.52	1.16E+01	-tan(0.688*omega_0/(-omega - 0.979*omega_0**2/omega) - 20.04) - 2.719
90	III.10.19	4.43E-01	mom*(0.59*Bx - 8.83*By/(-9.9 - 14.313*Bz/By) + 0.819*Bz)
91	III.12.43	3.05E-16	0.051*pi*h*n
92	III.13.18	5.05E-13	12.566*E_n*d**2*k/h
93	III.14.14	6.22E+00	I_0*(1.211*exp(tan(1.998*q)) + 1.044)
94	III.15.12	3.16E-16	2.0*U*(1 - cos(d*k))
95	III.15.14	3.32E-17	0.013*h**2/(E_n*d**2)
96	III.15.27	7.50E-16	6.283*alpha/(d*n)
97	III.17.37	2.86E-01	-0.848*alpha*bet*cos(1.135*theta + 21.559) + 0.252 + 1.021*bet/alpha
98	III.19.51	none	
99	III.21.20	1.23E-15	-1.0*A_vec*q*rho_c_0/m

References

[1] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017.

[2] S. Džeroski, P. Langley, and L. Todorovski, "Computational discovery of scientific knowledge," in *Computational Discovery of Scientific Knowledge*, Springer, 2007, pp. 1–14.

[3] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[4] A. P. Parkes, *A Concise Introduction to Languages and Machines*. Springer, 2008.

[5] L. Todorovski and S. Džeroski, "Declarative bias in equation discovery," in *Proceedings of the Fourteenth International Conference on Machine Learning, ICML*, Morgan Kaufmann, 1997, pp. 376–384.

[6] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[7] P. D. Grünwald, *The Minimum Description Length Principle*. MIT press, 2007.

[8] R. Guimerà, I. Reichardt, A. Aguilar-Mogas, *et al.*, "A Bayesian machine scientist to aid in the solution of challenging scientific problems," *Science Advances*, vol. 6, no. 5, eaav6971, 2020.

[9] J. Tanevski, L. Todorovski, and S. Džeroski, "Combinatorial search for selecting the structure of models of dynamical systems with equation discovery," *Engineering Applications of Artificial Intelligence*, vol. 89, p. 103 423, 2020.

[10] A. Katok and B. Hasselblatt, *Introduction to the Modern Theory of Dynamical Systems*. Cambridge University Press, 1995.

[11] W. Bridewell and P. Langley, "Two kinds of knowledge in scientific discovery," *Topics in Cognitive Science*, vol. 2, no. 1, pp. 36–52, 2010.

[12] W. Bridewell, P. Langley, L. Todorovski, and S. Džeroski, "Inductive process modeling," *Machine Learning*, vol. 71, pp. 1–32, 2008.

[13] S.-M. Udrescu and M. Tegmark, "AI Feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, no. 16, 2020.

[14] B. K. Petersen, M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," *arXiv:1912.04871*, 2019.

[15] K. Tashkova, "Parameter identification in nonlinear dynamic systems with metaheuristic approaches," PhD thesis, Jožef Stefan International Postgraduate School, Ljubljana, 2012.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[17] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[18] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. SIAM, 2019.

[19] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. John Wiley & Sons, 2006.

[20] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[21] P. Langley, "Data-driven discovery of physical laws," *Cognitive Science*, vol. 5, no. 1, pp. 31–54, 1981.

[22] B. W. Koehn and J. M. Zytkow, "Experimenting and theorizing in theory formation," in *Proceedings of the ACM SIGART International Symposium on Methodologies for Intelligent Systems*, 1986, pp. 296–307.

[23] B. C. Falkenhainer and R. S. Michalski, "Integrating quantitative and qualitative discovery: The abacus system," *Machine Learning*, vol. 1, no. 4, pp. 367–401, 1986.

[24] S. Dzeroski and L. Todorovski, "Discovering dynamics: From inductive logic programming to machine discovery," *Journal of Intelligent Information Systems*, vol. 4, no. 1, pp. 89–108, 1995.

[25] R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg, "Dendral: A case study of the first expert system for scientific hypothesis formation," *Artificial Intelligence*, vol. 61, no. 2, pp. 209–261, 1993.

[26] D. Čerepnalkoski, "Process-based models of dynamical systems: Representation and induction," PhD thesis, Jožef Stefan International Postgraduate School, Ljubljana, 2013.

[27] J. Tanevski, N. Simidjievski, L. Todorovski, and S. Džeroski, "Process-based modeling and design of dynamical systems," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2017, pp. 378–382.

[28] M. M. Kokar, "Determining arguments of invariant functional descriptions," *Machine Learning*, vol. 1, no. 4, pp. 403–422, 1986.

[29] T. Washio and H. Motoda, "Discovery of first-principle equations based on scale-type-based and data-driven reasoning," *Knowledge-Based Systems*, vol. 10, no. 7, pp. 403–411, 1998.

[30] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992, vol. 1.

[31] S. Sun, R. Ouyang, B. Zhang, and T.-Y. Zhang, "Data-driven discovery of formulas by symbolic regression," *MRS Bulletin*, vol. 44, no. 7, pp. 559–564, 2019.

[32] R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'neill, "Grammar-based genetic programming: A survey," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 365–396, 2010.

[33] C. Ryan, J. J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *European Conference on Genetic Programming*, Springer, 1998, pp. 83–96.

[34] G. Martius and C. H. Lampert, "Extrapolation and learning equations," *arXiv preprint arXiv:1610.02995*, 2016.

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[36] B. K. Petersen, C. P. Santiago, and M. Landajuela, "Incorporating domain knowledge into neural-guided search via in situ priors and constraints," Lawrence Livermore National Lab. Livermore, CA (United States), Tech. Rep., 2021.

[37] L. Crochepierre, L. Boudjeloud-Assala, and V. Barbesant, "A reinforcement learning approach to domain-knowledge inclusion using grammar guided symbolic regression," *arXiv preprint arXiv:2202.04367*, 2022.

[38] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *International Conference on Machine Learning*, 2017, pp. 1945–1954.

[39] S. Mežnar, S. Džeroski, and L. Todorovski, "Efficient generator of mathematical expressions for symbolic regression," *Machine Learning*, vol. 112, pp. 4563–4596, 4 2023.

[40] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017.

[41] M. Valipour, B. You, M. Panju, and A. Ghodsi, "SymbolicGPT: A generative transformer model for symbolic regression," *arXiv:2106.14131*, 2021.

[42] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end symbolic regression with transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 10 269–10 281, 2022.

[43] A. Ratle and M. Sebag, "Grammar-guided genetic programming and dimensional consistency: Application to non-parametric identification in mechanics," *Applied Soft Computing*, vol. 1, no. 1, pp. 105–118, 2001.

[44] F. O. de França, "A greedy search tree heuristic for symbolic regression," *Information Sciences*, vol. 442, pp. 18–32, 2018.

[45] E. Buckingham, "On physically similar systems," *Physical Review*, vol. 4, no. 4, p. 345, 1914.

[46] M. Keijzer and V. Babovic, "Dimensionally aware genetic programming," in *1st Annual Conference on Genetic and Evolutionary Computation*, vol. 2, 1999, pp. 1069–1076.

[47] V. Tsoutsouras, S. Willis, and P. Stanley-Marbell, "Deriving equations from sensor data using dimensional function synthesis," *Communications of the ACM*, vol. 64, no. 7, pp. 91–99, 2021.

[48] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, "AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4860–4871, 2020.

[49] M. Sipser, "Introduction to the theory of computation," *ACM SIGACT News*, vol. 27, no. 1, pp. 27–29, 1996.

[50] S. Geman and M. Johnson, "Probabilistic grammars and their applications," *International Encyclopedia of the Social & Behavioral Sciences*, vol. 2002, pp. 12 075–12 082, 2002.

[51] Z. Chi, "Statistical properties of probabilistic context-free grammars," *Computational Linguistics*, vol. 25, no. 1, pp. 131–160, 1999.

[52] A. Meurer, C. P. Smith, M. Paprocki, *et al.*, "Sympy: Symbolic computing in Python," *PeerJ Computer Science*, vol. 3, 2017.

[53] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media", 2009.

[54] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[55] Ž. Lukšič, J. Tanevski, S. Džeroski, and L. Todorovski, "Meta-model framework for surrogate-based parameter estimation in dynamical systems," *IEEE Access*, vol. 7, pp. 181 829–181 841, 2019.

[56] P. Lee, *Bayesian Statistics: An Introduction, 4th Edition*. John Wiley & Sons, 2012.

[57] C. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[58] S. G. Sterrett, "Physically similar systems – a history of the concept," in *Springer Handbook of Model-based Science*, Springer, 2017, pp. 377–411.

[59] G. I. Barenblatt, *Scaling*. Cambridge University Press, 2003.

[60] X. Shi, M. P. Brenner, and S. R. Nagel, "A cascade of structure in a drop falling from a faucet," *Science*, vol. 265, no. 5169, pp. 219–222, 1994.

[61] M. F. Modest and S. Mazumder, *Radiative Heat Transfer*. Academic Press, 1993.

[62] W. R. Stahl, "Dimensional analysis in mathematical biology," *The Bulletin of Mathematical Biophysics*, vol. 24, no. 1, pp. 81–108, 1962.

[63] A. Seminara, T. E. Angelini, J. N. Wilking, *et al.*, "Osmotic spreading of *Bacillus subtilis* biofilms driven by an extracellular matrix," *Proceedings of the National Academy of Sciences*, vol. 109, no. 4, pp. 1116–1121, 2012.

[64] M. Pohl, A. Ristig, W. Schachermayer, and L. Tangpi, "The amazing power of dimensional analysis: Quantifying market impact," *Market Microstructure and Liquidity*, vol. 3, p. 1 850 004, 2017.

[65] R. Kurth, *Dimensional Analysis and Group Theory in Astrophysics*. Elsevier, 2013.

[66] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end symbolic regression with transformers," *arXiv:2204.10532*, 2022.

[67] M. M. Kokar, "Determining arguments of invariant functional descriptions," *Machine Learning*, vol. 1, no. 4, pp. 403–422, 1986.

[68] T. Washio and H. Motoda, "Discovery of first-principle equations based on scale-type-based and data-driven reasoning," *Knowledge-Based Systems*, vol. 10, no. 7, pp. 403–411, 1998.

[69] M. Durasevic, D. Jakobovic, M. Scoczynski Ribeiro Martins, S. Picek, and M. Wagner, "Fitness landscape analysis of dimensionally-aware genetic programming featuring Feynman equations," in *Proceedings of the 16th International Conference on Parallel Problem Solving from Nature*, Springer, 2020, pp. 111–124.

[70] J. Bakarji, J. Callaham, S. L. Brunton, and J. N. Kutz, "Dimensionally consistent learning with Buckingham Pi," *arXiv:2202.04643*, 2022.

[71] L. Crochepierre, L. Boudjeloud-Assala, and V. Barbesant, "A reinforcement learning approach to domain-knowledge inclusion using grammar guided symbolic regression," *arXiv:2202.04367*, 2022.

[72] P. Deransart and M. Jourdan, "Attribute grammars and their applications," in *Proceedings of the International Conference WAGA*, vol. 461, Springer, 1990.

[73] J. Brence, L. Todorovski, and S. Džeroski, "Probabilistic grammars for equation discovery," *Knowledge-Based Systems*, vol. 224, p. 107 077, 2021.

[74] J. Vonn Neumann, "Various techniques used in connection with random digits," *National Bureau Standards*, vol. 12, pp. 36–38, 1951.

[75] I. J. Good, *Probability and the weighing of evidence*. Charles Griffing, 1950.

[76] I. J. Good, *The Estimation of Probabilities*. MIT Press, 1965.

[77] B. Cestnik, "Estimating probabilities: A crucial task in machine learning," in *Proceedings of the 9th European Conference on Artificial Intelligence*, 1990, pp. 147–149.

[78] B. Cestnik and I. Bratko, "On estimating probabilities in tree pruning," in *Proceedings of Machine Learning – European Working Session on Learning 91*, Springer, 1991.

[79] S. Džeroski, B. Cestnik, and I. Petrovski, "Using the m-estimate in rule induction," *Journal of Computing and Information Technology*, vol. 1, no. 1, pp. 37–46, 1993.

[80] M. Chaushevska, L. Todorovski, J. Brence, and S. Džeroski, "Learning the probabilities in probabilistic context-free grammars for arithmetical expressions from equation corpora," in *Proceedings of the 25th International Multiconference Information Society*, Jožef Stefan Institute, Ljubljana, Slovenia, vol. A, 2022, pp. 11–14.

[81] T. S. C. Foundation. "ISO International Standard ISO/IEC 14882:2020(E) – Programming Language C++." (2020), [Online]. Available: `https://isocpp.org/std/the-standard`.

[82] R. Team. "Rust programming language." (2024), [Online]. Available: `https://www.rust-lang.org/`.

[83] O. Fajardo-Fontiveros, I. Reichardt, H. R. De Los Rios, J. Duch, M. Sales-Pardo, and R. Guimerà, "Fundamental limits to learning closed-form mathematical models from data," *Nature Communications*, vol. 14, no. 1, p. 1043, 2023.

Bibliography

Publications Related to the Thesis

Journal Articles

J. Brence, L. Todorovski, and S. Džeroski, "Probabilistic grammars for equation discovery," *Knowledge-Based Systems*, vol. 224, p. 107 077, 2021.

J. Brence, S. Džeroski, and L. Todorovski, "Dimensionally-consistent equation discovery through probabilistic attribute grammars," *Information Sciences*, vol. 632, pp. 742–756, 2023.

Conference Paper

B. Gec, N. Omejc, J. Brence, S. Džeroski, and L. Todorovski, "Discovery of differential equations using probabilistic grammars," in *Proceedings of the 25th International Conference on Discovery Science*, Springer, 2022, pp. 22–31.

M. Chaushevska, L. Todorovski, J. Brence, and S. Džeroski, "Learning the probabilities in probabilistic context-free grammars for arithmetical expressions from equation corpora," in *Proceedings of the 25th International Multiconference Information Society*, Jožef Stefan Institute, Ljubljana, Slovenia, vol. A, 2022, pp. 11–14.

Other Publications

Journal Articles

J. Brence, L. Cmok, N. Sebastián, A. Mertelj, D. Lisjak, and I. Drevenšek-Olenik, "Optical second harmonic generation in a ferromagnetic liquid crystal," *Soft Matter*, vol. 15, no. 43, pp. 8758–8765, 2019.

J. Brence, J. Tanevski, J. Adams, E. Malina, and S. Džeroski, "Surrogate models of radiative transfer codes for atmospheric trace gas retrievals from satellite observations," *Machine Learning*, vol. 112, no. 4, pp. 1337–1363, 2023.

J. Brence, D. Mihailović, V. V. Kabanov, L. Todorovski, S. Džeroski, and J. Vodeb, "Boosting the performance of quantum annealers using machine learning," *Quantum Machine Intelligence*, vol. 5, no. 1, p. 4, 2023.

Biography

Jure Brence was born on 25. October 1993 in Ljubljana. He finished his primary education at "II. OŠ Rogaška Slatina" and his secondary education at "Šolski center Rogaška Slatina", program "Splošna gimnazija". He was awarded the golden award for his results at the national exam (matura) in 2012 and began his study of Physics at the Faculty of Mathematics and Physics (FMF), University of Ljubljana, in the same year.

During his second year of bachelor's studies, he was involved in research at the Jožef Stefan Institute (JSI) as a student at the Department of Intelligent Systems under the supervision of Prof. Dr. Matjaž Gams. During his third and the additional fourth year, he was closely involved in research work at the ultracold atoms laboratory, Department of Condensed Matter Physics (JSI) under the supervision of Dr. Peter Jeglič.

After completing his bachelor's degree in 2016, he continued his studies of Technical Physics and Photonics (FMF) at the master's level. He took part in an Erasmus student exchange in Vienna, Austria. During the second and the additional third year of studies, he was intensively involved in research work in the optics group at the Department of Complex Matter (JSI) under the supervision of Prof. Dr. Irena Drevenšek. The research culminated with the defense of his Master's thesis, titled "Second harmonic generation in feromagnetic liquid crystals". The thesis was awarded the faculty Prešeren prize.

In 2019, he enrolled in the PhD study program Information and Communication Technologies at the Jožef Stefan International Postgraduate School and started working as a research assistant at the Department of Knowledge Technologies (JSI) under the supervision of Prof. Dr. Sašo Džeroski and Prof. Dr. Ljupčo Todorovski.

His research interests lie in the area of machine learning, especially its use in physics and other natural sciences, with a focus on methods of equation discovery. He has published several scientific papers, presented his work at international conferences and taken part in summer and winter schools on related topics.