ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Seminar for Applied Mathematics SAM

The Language of Hyperelastic Materials

G. Kissas and S. Mishra and E. Chatzi and L. De Lorenzis

Research Report No. 2024-07
January 2024

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

THE LANGUAGE OF HYPERELASTIC MATERIALS

Georgios Kissas
AI Center
ETH Zurich
gkissas@ai.ethz.ch

Siddhartha Mishra
Seminar for Applied Mathematics
ETH Zurich
siddhartha.mishra@sam.math.ethz.ch

Eleni Chatzi
Department of Civil, Environmental and Geomatic Engineering
ETH Zurich
chatzi@ibk.baug.ethz.ch

Laura De Lorenzis
Department of Mechanical and Process Engineering
ETH Zurich
ldelorenzis@ethz.ch

ABSTRACT

The automated discovery of constitutive laws forms an emerging area that focuses on automatically obtaining symbolic expressions describing the constitutive behavior of solid materials from experimental data. Existing symbolic/sparse regression methods rely on availability of libraries of material models, which are typically hand-designed by a human expert relying on known models as reference, or deploy generative algorithms with exponential complexity which are only practicable for very simple expressions. In this paper, we propose a novel approach to constitutive law discovery relying on formal grammars as an automated and systematic tool to generate constitutive law expressions complying with physics constraints. We deploy the approach for two tasks: i) Automatically generating a library of valid constitutive laws for hyperelastic isotropic materials; ii) Performing data-driven discovery of hyperelastic material models from displacement data affected by different noise levels. For the task of automatic library generation, we demonstrate the flexibility and efficiency of the proposed methodology in alleviating hand-crafted features and human intervention. For the data-driven discovery task, we demonstrate the accuracy, robustness and significant generalizability of the proposed methodology.

***K*eywords** Automated Model Discovery · Data-Driven Constitutive Models · Formal Grammars · Symbolic Regression · Generative AI

1 Introduction

From the mechanics of a beating heart to the haptics of a robotic arm, from the rupture of an aneurysm sack to the aeroelasticity of a spaceship, mechanical phenomena controlled by the behavior and properties of different types of materials are ubiquitous in nature and in engineering applications alike. Thus, understanding, modeling and characterizing the mechanical response of materials has been an important research focus for centuries. Accordingly, extracting so-called material models (or constitutive laws), i.e. relations between stresses, strains and often additional variables, from experimental data has been and still is a central task in solid mechanics [1]. The conventional approach to solve this underlying inverse problem requires a large number of experiments and a tedious trial-and-error iterative procedure, involving at each iteration the choice of a model from the available literature (largely based on experience) and the subsequent calibration of its unknown parameters [2], often referred to as model updating, which can be achieved under both deterministic and stochastic schemes [3, 4]. However, the recent advances in imaging techniques

are increasingly motivating a paradigm shift from point-wise measurements to full-field data acquisition [5]. At the same time, the stunning recent successes of machine learning, and more generally data-driven techniques, have spawned a dramatic surge in the adoption of such algorithms also in the field of material modeling [6]; see [7] for a recent review. In this work, we leverage machine learning techniques to automatize model selection and calibration by condensing these into the single task of model discovery, see the difference in model identification and discovery in [2]. Moreover, we are interested in approaches which deliver as output an interpretable material model, i.e. one that can be represented by mathematical expressions satisfying appropriate constraints, thus moving away from the model-free paradigm [8].

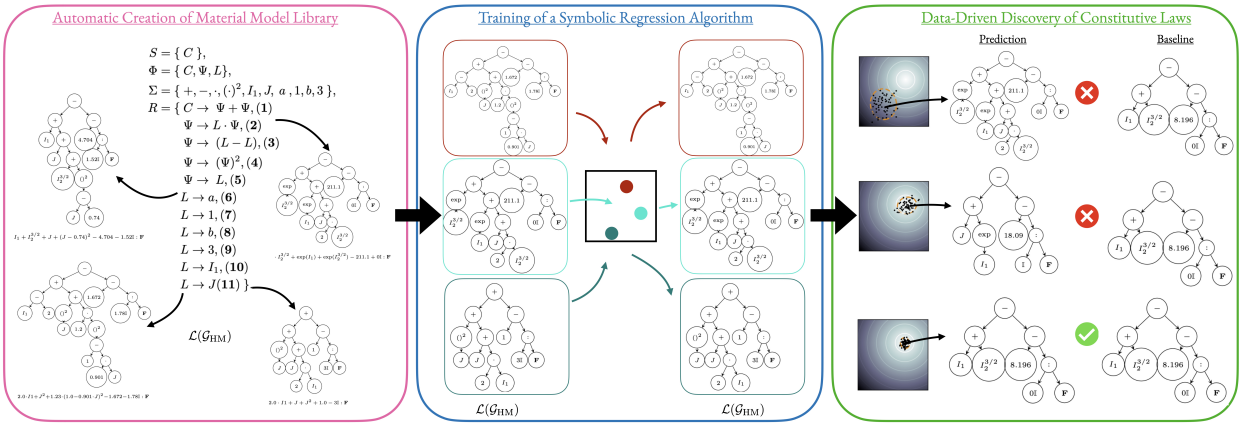


Figure 1: A graphical representation of the process proposed in this manuscript. In a first step, a virtual material library is assembled offline using a formal grammar. In a second step, a symbolic regression algorithm is trained offline on the generated virtual material library. In a third step, constitutive laws are discovered from experimental data using gradient-free optimization, and the trained model from the second step, for solving the resulting inverse problem.

In general, approaches that derive mathematical expressions from targeted experimental data are commonly referred to as symbolic regression methods. In what follows, we review different symbolic regression methods proposed in the literature for general (i.e. not necessarily material) model discovery and comment on their advantages and shortcomings in relation to the discovery of constitutive laws. For the sake of a unified presentation, we choose to view symbolic regression through the lens of operations on graphs. We further refer readers not familiar with the graph representation of mathematical expressions to Appendix A, which reviews some basic notions that facilitate reading. From this perspective, symbolic regression algorithms can be viewed as structured ways to add and discard edges of a graph between nodes, with the nodes containing so-called primitives (i.e. constants, variables, mathematical operations, or small expressions). Thus, the construction of a symbolic regression method entails three main choices – i) how the primitives are chosen; ii) what is the underlying graph topology; iii) how to ensure the derivation of meaningful expressions for the chosen graph topology. We hereby discuss existing classes of such algorithms in terms of these defining choices.

Equation Learner. A simple and interpretable framework for symbolic regression is the Equation Learner (EQL) [9]. To construct the EQL algorithm, the primitives include mathematical operations, such as the $\sin$, $\cos$, identity and $\cdot$ operations, and the input variable X, whereas the underlying topology is a weighted directed graph topology that represents all the functions that provide an output y given an input X, see Figure 2. This is equivalent to considering a fully connected neural network whose weights are the weights of the graph and whose activation functions are the primitive operations. Multiple layers of the network are considered for approximating complex functions and different paths of the graph represent different mathematical expressions. The method is trained such that, given an input, it determines the edges of the graph that constitute the simplest path whose resulting expression evaluation is the closest to the measurements. Even though this method has been successfully employed for different tasks, it comes with some important drawbacks. First, there is no systematic way to constrain the method to provide meaningful mathematical expressions. Moreover, EQL is difficult to train for primitives such as division, exponentiation and logarithms even though ways to alleviate this issue have been proposed [10, 11]. Since these operations are present in many material models, the application of this method to constitutive law discovery can be problematic. An EQL-based approach for discovery of hyperelastic constitutive laws is presented in [12, 13]; the authors propose methods that consider the invariants of the Cauchy-Green tensor as inputs and activation functions consisting of unary operations. These authors do not consider constitutive laws containing operations such as division, exponentiation or logarithms.

Sparse Regression. An alternate family of algorithms is based on Sparse Regression (SR) [14, 15, 16]. For constructing a SR algorithm, primitives are chosen to be expressions, rather than operations, and the underlying topology is that of

a weighted directed acyclic graph connecting the root node (i.e. the input X) with the primitives, see Figure 2. This is equivalent to considering a fully connected network without activation functions, trained to provide the weighted combination of edges that best fits the given measurements, while also imposing sparsity [17] with the purpose of obtaining a parsimonious, i.e. a simple mathematical expression. In SR, whether a generated expression is meaningful is (at least partially) determined by the primitives. For example, considering simple constitutive laws already as primitives is likely to result in a valid final constitutive law. This approach has been successfully applied in different areas of scientific discovery [18] and also to the discovery of constitutive laws with applications to hyperelasticity [2, 19, 20, 21, 22, 23], viscoelasticity [24], plasticity [25, 26] and generalized standard material models [27]. The main drawback of SR is that a human expert needs to hand-design the primitives; this inevitably introduces a bias in the process of discovery and restricts the search space of possible candidate expressions to those included in the starting library.

Genetic Programming. Instead of imposing sparsity to a directed acyclic graph, other methods achieve sparsity by operating directly on trees rather than graphs; in other words, sparsity naturally follows from the tree structure. A standard approach for performing symbolic regression on trees is Genetic Programming (GP) [28, 29]. Here, an initial population of S-expressions is constructed randomly by combining different primitives, i.e. constants as well as unary or binary operators [30]. For each random generation, actions are performed that alter step-by-step the structure of the population of the tree, by either mutating the primitives or removing/adding sub-expressions to the tree. This approach is not equipped with any structured way to impose constraints to each step of the generation of mathematical expressions. Moreover, the complexity of the method grows very fast with the depth of the tree [31], making it practically applicable only to small mathematical expressions. The prohibitive computational cost for potentially complex expressions makes GP not suitable for constitutive law discovery. Nevertheless, attempts in this direction have been made [32, 33, 34, 26]. In these cases, initial expression trees are evolved using mutations and crossover operations either as a standalone procedure or as a part of a more general pipeline.

Deep Symbolic Regression. Deep Symbolic Regression (DSR) [35] exploits Recurrent Neural Networks to predict the probability of children nodes given the parent node. To construct the DSR algorithm one needs to choose a set of primitives, namely constants as well as unary and binary operations, and a specific sequence of operations that corresponds to a top-to-bottom, left-to-right ordering of tree nodes, see Figure 2. The algorithm randomly chooses a root node; then, using auto-regressive sampling, it computes the probability of the next primitive on the tree conditioned on the previous one. The model is trained using a risk-seeking policy gradient to generate best-fitting expressions with high probability. In-situ constraints to zero out the probability of particular children given the parents can be considered to shrink the search space [35, 17, 36]. The difficulty in applying this methodology to a constitutive law discovery scenario (which, to the best of our knowledge, has not yet been attempted) is that the constraints are imposed at each step of the generation but not to the whole expression. Moreover, the generation process requires a second optimization step to calibrate constants, which may end up violating constitutive law constraints.

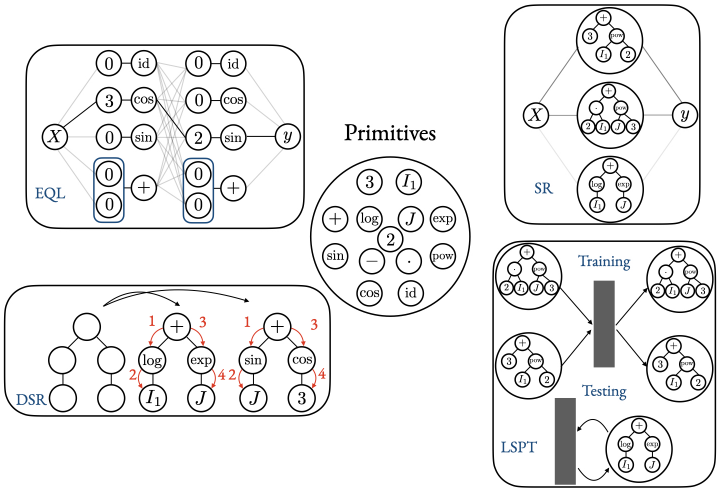


Figure 2: Schematic representation of abstractions of different symbolic regression methods. For EQL and SR, we are performing supervised learning to learn a map from X to y parameterized by a network, while also imposing sparsity to the network. After the training is completed, we track the remaining edges to discover the form of the equation. The DSR and LSPT methods are constructed by randomly generating and evaluating trees based on a set of primitives, so they do not directly involve supervised learning.

Large-Scale Pre-Trained Models. The approaches we introduced so far, learn a model that needs to be re-trained as new data points are added. To address this issue in symbolic regression, Large-Scale Pre-Trained (LSPT) models have been proposed [37, 38, 39]. LSPT models are trained once on a library of *valid* symbolic expressions and then provide the probability of an expression given a new data point. Constructing a LSPT model entails choosing primitives as a set of constants and mathematical operations, and a tree topology with a sequence of operations that corresponds to a top-to-bottom, left-to-right ordering of the tree nodes. By sampling *valid* sequences of primitives, a library of potential expressions is constructed and then evaluated for a range of the input variable X and constant values. The pair consisting of the input variable X and the evaluation of the expression y is encoded using a transformer architecture and the encoding is then used by a different transformer architecture to provide a distribution of expressions. The whole process is trained end-to-end by supervised learning via matching the function evaluation and the tree labels. After the model is trained, a beam search is performed for pairs of (X, y) to discover expressions that provide y with high likelihood, see Figure 2. This method could potentially translate to constitutive law discovery, but it would need to be pre-trained on a large library of expressions. At present, there exists no systematic way to construct such library, nor to impose constraints during the library creation or the expression generation process.

Grammars and Variational Autoencoders. Formal grammars have been explored for the generation of molecules as well as of mathematical expressions using sequence-to-sequence [40, 41] and recursive [42] learning. Discovery processes of both new molecules and mathematical expressions can be succeeded via use of a deep generative model, more specifically using a variational autoencoder (VAE). In Kusner et. al. [40], the authors propose a method that first extracts from a tree the sequence of grammar rules that generates it and then encodes the sequence using a low-dimensional vector. In Paassen et. al. [42], the authors directly encode a tree using a recursive procedure that encodes the rule that generated the parent node given the children. In both cases, a decoding algorithm based on a grammar is considered to guarantee syntactic validity of the expressions. After the VAE model is trained, a gradient-free optimization procedure, such as Bayesian optimization or an evolutionary strategy, can be employed for exploring the low-dimensional space of the VAE to find the expression that best fits a new set of measurements. Methods based on grammar possess low complexity, constrain the tree generation using rules, and generalize. To the best of our knowledge, they have never been used for constitutive law discovery.

The above survey on available tools for symbolic regression clearly suggests the following characteristics that would be desirable for symbolic regression algorithms applied to data-driven constitutive law discovery:

- Accommodation of primitive operations of significant complexity (e.g. including $\exp$, $\log$ or $/$), without exhibiting training instabilities.
- Alleviation of human bias via flexible and automated generation of new constitutive expressions.
- Low complexity, meaning that the complexity of the discovery process should not increase exponentially with the length of the expression.
- Possibility to incorporate constraints on both the final expression and the steps of the expression generating process.
- Generalization of the symbolic regression algorithm, i.e. potential to provide expressions that are valid for new and unseen data points without re-training.

In this paper, we propose a symbolic regression pipeline for constitutive law discovery that satisfies all the above-listed requirements. Due to the advantages of formal grammars mentioned earlier (low complexity, ability to systematically embed constraints stemming from domain knowledge, and generalization capability) we explore grammar-based symbolic regression and develop formal grammars which are specifically designed for constitutive laws. In this first investigation, we focus on hyperelasticity, whereby the material behavior is fully encoded by the elastic strain energy density function for which we seek a parsimonious mathematical expression.

The proposed pipeline is composed by three steps: a library construction step, a pre-training step, and a data-driven discovery step, as illustrated in Figure 1. In the first step, we define a formal grammar that generates mathematical expressions in the form of trees. We ensure that the expressions derived by the grammar are valid constitutive laws (i.e., in our case, valid elastic strain energy density functions) by applying constraints either during the generation process or to the final expressions. The grammar is then used to perform an off-line library generation of constitutive laws. Note that this library is a useful result per se, as it may be deployed in place of a hand-crafted library in the context of constitutive model discovery based on sparse regression [2]. In the second step, this library is used to train off-line a symbolic regression algorithm that takes advantage of the tree structure of the mathematical expressions and of the grammar of the constitutive laws. The pre-trained model is trained to encode a tree to a low-dimensional latent vector representations and decode this low-dimensional vector representation to a tree, creating a low-dimensional manifold of tree representations. In the third step, we perform a gradient-free optimization approach to search the low-dimensional manifold for the vectorial representation of the model that best fits the given measurements. Therefore,

the expensive step of training the deep learning model needs to be executed once and then we perform the discovery without re-training.

The remainder of the manuscript is structured as follows. In Section 2 we focus on the notion of formal grammars and describe the Context-Free and Regular Tree Grammar classes, their properties and semantics. Section 3 is devoted to the construction of a formal language, the Language of Hyperelastic Materials, where the constitutive law constraints are either included in the grammar or enforced on the derived expressions. In this section we also discuss how the Language of Hyperelastic Materials can deployed for the automatic construction of a library of hyperelastic constitutive laws, and present examples of generation of such a library. In Section 4, we introduce a symbolic regression method combining VAEs and a Regular Tree Grammar defined for constitutive laws, that fulfils all the requirements of the earlier list. Finally, we propose a pipeline for discovering constitutive laws from available data and illustrate an example using artificially generated full-field displacement data. Conclusions and an outlook close the paper in Section 5.

2 Context-Free and Regular Tree Grammars

In this section, we present two different types of formal grammars, namely Context-Free Grammars and Regular Tree Grammars, along with a simple example to showcase their use. To enable a high-level intuition of how grammar works, we provide a simplified schematic representation of a tree created using the proposed grammar in Figure 3. The node labels are variables, e.g. L, Ψ, S, from which one, in this case S, is assigned as the tree root. During tree generation, these variables are substituted with different primitives (i.e. constants, e.g. 2, 3, variables, e.g. I_1, J, or operators, e.g. $+, -, \cdot$), using predefined substitution rules, e.g. r_1, r_2. The number of arguments that the primitives take (e.g. $+$ takes two arguments while J takes no arguments) determine the number of their children and the tree connectivity.

2.1 Context-Free Grammars

Roughly described, a Context-Free Grammar (CFG) is a systematic way of generating tree structures that represent syntactically and semantically meaningful expressions using a set of rules. CFGs are defined as a tuple $\mathcal{G} = \{\Phi, \Sigma, R, S\}$, where Φ is a set of non-terminal symbols, Σ is an alphabet of terminal symbols, R is a set of production rules, and S denotes a special non-terminal symbol called the starting symbol. Let us explain the meaning of these terms.

- *Non-terminal* symbols are the variable node labels (i.e. L, Ψ, S in Figure 3). They are syntactic variables that cannot appear in an expression as standalone entities; to generate a sentence, or, in our context, a mathematical expression, they are substituted by terminal symbols through production rules.

- *Terminal symbols* are the primitives. They are the building blocks that compose sentences or, in our context, mathematical expressions. In our case, the terminal symbols can be defined as constants, variables, operations (such as in Figure 3), or sub-expressions.

- *Production rules* are user-defined rules to substitute non-terminals with other non-terminal or terminal symbols. Each substitution rule consists of a left-hand side that contains a non-terminal symbol, and a right-hand side that contains a mixture of terminals and non-terminals to be substituted to the left-hand side. E.g., the rule $\Psi \rightarrow sL$, with $\Psi, L \in \Phi$ and $s \in \Sigma$, substitutes Ψ with sL in a sentence. If a non-terminal symbol appears on the left-hand side of more than one production rule, it can be replaced by any of the right-hand sides of these rules. By recursive substitution of the non-terminal symbols using the production rules, initiating from the starting symbol S, sentences that contain only terminal symbols are finally derived.

Note that because the production rules of CFGs contain only one non-terminal on the left-hand side, the sentences of CFGs can be expressed as trees, called derivation trees, as illustrated in Figure 3. Each derivation tree corresponds to a terminal sentence, or, in our context, to a mathematical expression. A language $\mathcal{L}(\mathcal{G})$ is defined as the set of all possible terminal sentences that can be derived by applying the production rules of the grammar starting from S, or all possible ways that the nodes of a derivation tree can be connected starting from S.

We now illustrate these concepts with a simple example. Consider the grammar $\mathcal{G}_{\text{NH}} = \{\Phi, \Sigma, R, S\}$, where:

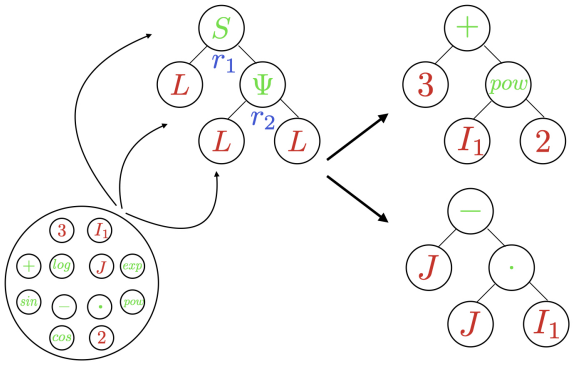


Figure 3: Schematic explanation of a tree derivation using grammar. The non-terminals are color coded based on the grammar rules that substitute them. For example, L is substituted by 3, I_1, or J and is re-written by the grammar rules $J \to 3|I_1|J$. On the other hand, Ψ and S are re-written by $+$, $-$, $\cdot$, or pow, and correspond to $S \to +(L, \Psi)| - (L, \Psi)$, and $\Psi \to pow(L, L)| \cdot (L, L)$, respectively. The black arrows show two possible tree derivation generated by these substitutions.

$$
\begin{aligned}
S &= \{\ C\ \}, \\
\Phi &= \{\ C, \Psi, L\}, \\
\Sigma &= \{\ +, -, \cdot, (\cdot)^2, I_1, J,\ a\ , 1, b, 3\ \}, \\
R &= \{\ C \to\ \Psi + \Psi, (\mathbf{1}) \\
&\qquad \Psi \to L \cdot \Psi, (\mathbf{2}) \\
&\qquad \Psi \to\ (L - L), (\mathbf{3}) \\
&\qquad \Psi \to\ (\Psi)^2, (\mathbf{4}) \\
&\qquad \Psi \to\ L, (\mathbf{5}) \\
&\qquad L \to a, (\mathbf{6}) \\
&\qquad L \to 1, (\mathbf{7}) \\
&\qquad L \to b, (\mathbf{8}) \\
&\qquad L \to 3, (\mathbf{9}) \\
&\qquad L \to I_1, (\mathbf{10}) \\
&\qquad L \to J (\mathbf{11})\ \}.
\end{aligned}
$$

Here, C, Ψ, and L denote the non-terminals, with C as the starting symbol, and Ψ and L are used for recursive substitution of operations and literals (i.e. constants and variables), respectively. More specifically, Ψ describes both unary and binary operations between terminals and non-terminals, whereas L re-writes the integers $1, 3$, the real constants a, c and the variables I_1, J. The numbers next to the production rules are used for their annotation. Note that rule $(\mathbf{3})$ is written as $(L - L)$, but this does not mean that this rule always provides zero values. Each non-terminal L is treated separately, which means that the two L's in rule $(\mathbf{3})$ do not always have the same value.

If we identify I_1, J with the first invariant of the right Cauchy-Green deformation tensor and the third invariant of the deformation gradient in finite deformation kinematics, we can produce the non-dimensional elastic strain energy density function of a simple Neo-Hookean model $W = a \cdot (I_1 - 3) + b \cdot (J - 1)^2$ by performing recursive substitutions of

non-terminal symbols beginning from the starting symbol C, as follows:

$$
\begin{aligned}
C &\xrightarrow{(\mathbf{1})} \Psi + \Psi, \\
&\xrightarrow{(\mathbf{2})} L \cdot \Psi + L \cdot \Psi, \\
&\xrightarrow{(\mathbf{3})} L \cdot (L - L) + L \cdot (L - L), \\
&\xrightarrow{(\mathbf{6})} a \cdot (L - L) + L \cdot (L - L), \\
&\xrightarrow{(\mathbf{10})} a \cdot (I_1 - L) + L \cdot (L - L), \\
&\xrightarrow{(\mathbf{9})} a \cdot (I_1 - 3) + L \cdot (L - L), \\
&\xrightarrow{(\mathbf{8})} a \cdot (I_1 - 3) + b \cdot (L - L), \\
&\xrightarrow{(\mathbf{11})} a \cdot (I_1 - 3) + b \cdot (J - L), \\
&\xrightarrow{(\mathbf{7})} a \cdot (I_1 - 3) + b \cdot (J - 1).
\end{aligned}
$$

The grammar $\mathcal{G}_{\text{NH}}$ can not only produce this specific Neo-Hookean model, but also generate any other expression that can be derived as a combination of the introduced production rules $\{(\mathbf{1}), ...(\mathbf{11})\}$. Examples derived from $\mathcal{G}_{NH}$ are shown in Figure 4. All the possible terminal strings derived by recursively substituting the non-terminal symbols $C, \Psi, L \in \Phi$ (starting from C) using the production rules constitute the context-free language $\mathcal{L}(\mathcal{G}_{\text{NH}})$.

2.2 Regular Tree Grammars

Another type of grammar suitable for our application is a Regular Tree Grammar (RTG), defined as the tuple $\mathcal{G} = \{\Phi, \tilde{\Sigma}, R, S\}$, where $\tilde{\Sigma}$ is now a ranked alphabet. This is defined as an alphabet augmented by specifying the arity of each primitive, i.e. the number of arguments each primitive takes. To explain the difference between CFGs and RTGs, we now consider a Regular-Tree version of $\mathcal{G}_{\text{NH}}$, which we denote as $\hat{\mathcal{G}}_{\text{NH}}$, where:

$$
\begin{aligned}
S &= \{\ C\ \}, \\
\Phi &= \{\ C, \Psi, L\}, \\
\tilde{\Sigma} &= \{\ +:2, -:2, \cdot:2, ()^2:1, I_1:0, J:0,\ a:0\ , 1:0, b:0, 3:0\ \}, \\
R &= \{\ C \rightarrow\ +(\Psi, \Psi), (\mathbf{1}) \\
&\qquad \Psi \rightarrow \cdot(L, \Psi), (\mathbf{2}) \\
&\qquad \Psi \rightarrow\ -(L, L), (\mathbf{3}) \\
&\qquad \Psi \rightarrow\ ()^2(\Psi), (\mathbf{4}) \\
&\qquad \Psi \rightarrow\ L, (\mathbf{5}) \\
&\qquad L \rightarrow a(), (\mathbf{6}) \\
&\qquad L \rightarrow 1(), (\mathbf{7}) \\
&\qquad L \rightarrow b(), (\mathbf{8}) \\
&\qquad L \rightarrow 3(), (\mathbf{9}) \\
&\qquad L \rightarrow I_1(), (\mathbf{10}) \\
&\qquad L \rightarrow J()(\mathbf{11})\ \}.
\end{aligned}
$$

There are two immediate differences between $\hat{\mathcal{G}}_{\text{NH}}$ and $\mathcal{G}_{\text{NH}}$. First, we provide the arity of each primitive in the alphabet. We consider operations with arity $= 2$, such as $+$, $-$, with arity $= 1$ such as $()^2$, and with arity $= 0$, i.e. constants and variables. Operations such as $()^2$ can also be defined to have arity $= 2$ in the general case where the exponent is also an argument. Second, we write all rules in Polish notation, see Section A, to denote that we are working with trees and not strings. As for CFGs, the sentences (or mathematical expressions) of RTGs are expressed as derivation trees. Also, as for the context-free case, the Regular-Tree Language $\mathcal{L}(\hat{\mathcal{G}})$ is defined as the set of all the trees that can be generated using the RTG $\hat{\mathcal{G}}$.

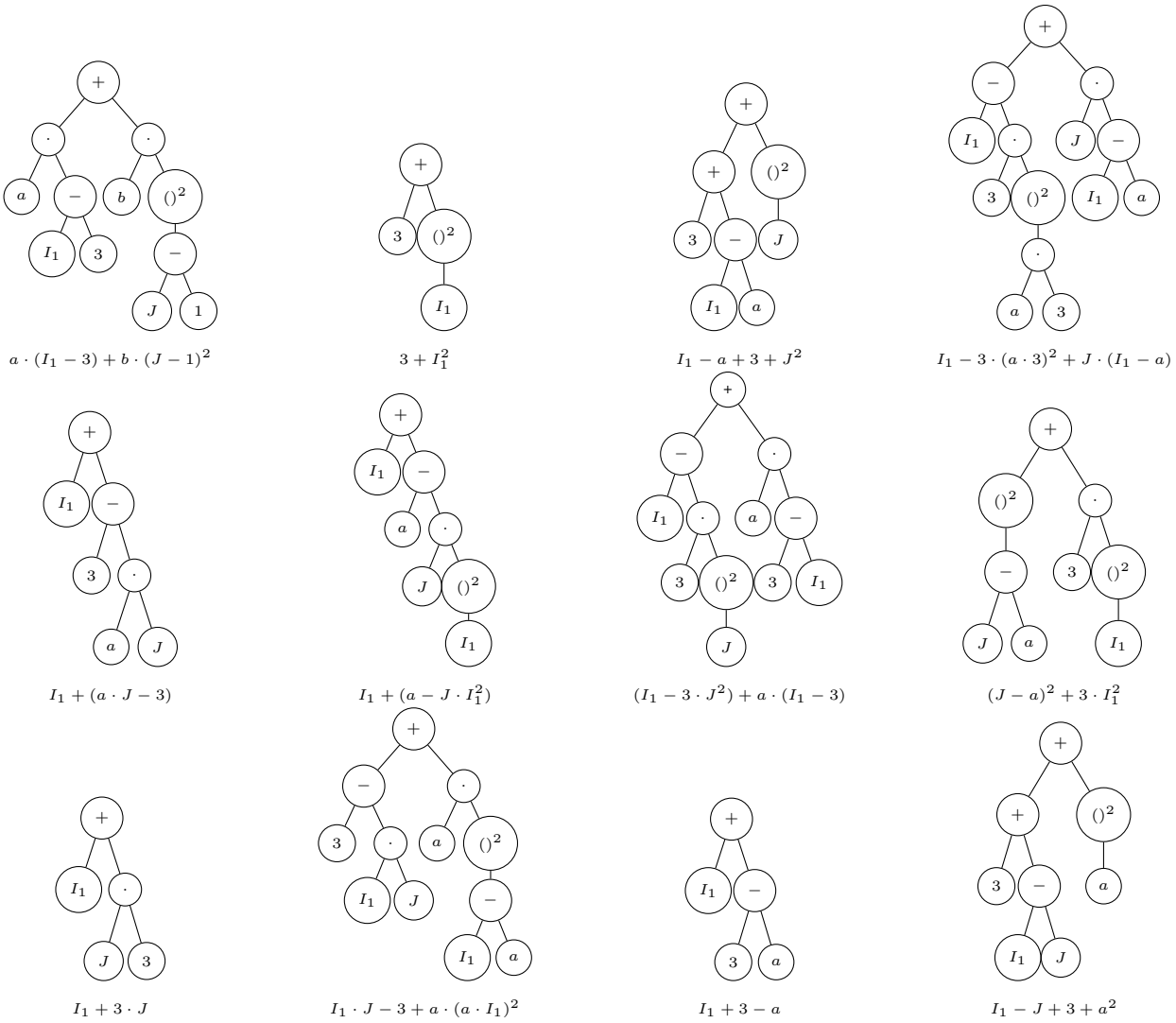


Figure 4: Binary trees derived from the context-free grammar $\mathcal{G}_{NH}$; they are all part of the language $\mathcal{L}(\mathcal{G}_{NH})$.

2.3 Characteristics and Operations of Tree Grammars

In general cases, trees are fully described by their node labels and connectivity. For the special case of a tree derived from a RTG, the tree can be fully characterized by the list of rules used for its generation, as we state more precisely in the following. In the previous section we have shown that a unique Neo-Hookean model can be derived by a set of rules. For the case of a RTG, the opposite also holds, meaning that all the information required to describe the specific Neo-Hookean model is uniquely represented by a set of rules. More precisely, it can be proven mathematically [42] that this property holds for unambiguous RTGs, i.e. RTGs in which each rule has a unique right-hand side. One can also prove that any RTG can be made unambiguous and that algorithms handling RTGs possess linear complexity in both computations and memory [42], see Appendix B for details on the properties of RTGs.

In view of the above, there are two meaningful actions to be performed on a tree derived from a RTG: i. given the tree, extract its characterization (i.e. the list of rules of the corresponding grammar used for its generation), which is denoted as *parsing*; and ii. given a characterization, i.e. a list of rules, generate the tree that it corresponds to, which is called *generation*. In Appendix B we provide details on tree parsing and generation algorithms. Note that knowledge on characterization is key for performing tree-based symbolic regression, as characterization (i.e. the list of rules) is the dependent variable that the symbolic regression algorithm learns to predict.

Importantly, the bijective relation between trees and the sets of grammar rules used for their generation only holds for (unambiguous) RTGs, and not for CFGs [43]. On the other hand, CFGs are known to be more expressive than

RTGs [44]. Looking ahead to the two main outcomes of this work, namely, automated generation of a material model library and automated model discovery based on data (Sections 3 and 4 respectively), it is quite clear that CFGs are more suitable for the first task, in which expressivity is important and the unique definition of the set of grammar rules corresponding to a given model is not essential, whereas RTGs are more suitable for the second task. For this reason, we will make use of CFGs in Section 3 and of RTGs in Section 4.

A grammar describes the rules that create sentences and constitute the syntax of natural languages. However, the syntax alone is not enough to define a language that is useful for our purposes; the additional needed ingredient is semantics, i.e. the meaning of the sentences that the language produces. While it is difficult to define semantics in natural languages, in physics this is more straightforward. For our purposes, *we consider as semantically valid the expressions that satisfy the constraints of the constitutive law relations*.

3 The Language of Hyperelastic Materials

In this section, our objective is to generate the Language of Hyperelastic Materials. We first review the constraints that need to be satisfied for a mathematical expression to represent a valid elastic strain energy density function. We then discuss ways of enforcing these constraints. This is done partially during the construction of the grammar and partially through checks on the final derived expressions. Subsequently, we propose a general language for hyperelastic materials, i.e. a language whose expressions are valid elastic strain energy density functions (i.e. functions which correspond to valid hyperelastic constitutive laws). Throughout this paper, we assume to have introduced a suitable non-dimensionalization such that we only deal with non-dimensional strain energy density functions.

We consider finite-deformation kinematics and denote with $\mathbf{u} \in \mathbb{R}^3$ the displacement field, with $\mathbf{F} = \mathbf{I} + \mathrm{Grad}\,\mathbf{u}$ the deformation gradient (where Grad is the gradient operator with respect to the reference coordinates), and with $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ the right Cauchy-Green deformation tensor. The principal invariants of $\mathbf{C}$ are defined as:

$$J = \det \mathbf{F} = (\det \mathbf{C})^{1/2}, \quad \mathrm{I}_1 = \mathrm{tr}\,\mathbf{C}, \quad \mathrm{I}_2 = \frac{1}{2}(\mathrm{tr}^2\,\mathbf{C} - \mathrm{tr}\,\mathbf{C}^2),$$

where tr denotes the trace operator. For nearly incompressible materials, it is customary to decompose the deformation gradient $\mathbf{F}$ into a volume-preserving (or isochoric) part, $\mathbf{F}^{iso}$, and a volume-altering part, $\mathbf{F}^{vol}$:

$$\mathbf{F} = \mathbf{F}^{iso}\mathbf{F}^{vol}, \tag{1}$$

where

$$\mathbf{F}^{iso} = J^{-1/3}\mathbf{F} \Rightarrow \quad \det \mathbf{F}^{\mathrm{iso}} = 1.$$

This decomposition translates to the right Cauchy-Green tensor and its invariants as follows

$$\mathbf{C}^{iso} = J^{-2/3}\mathbf{C}, \quad I_1^{iso} = \tilde{I}_1 = J^{-2/3}I_1, \quad I_2^{iso} = \tilde{I}_2 = J^{-4/3}I_2$$

With the decomposition of the deformation gradient, a frequent choice is to write the strain energy density function $W(\mathbf{F})$ as the sum of isochoric, W^{iso}, and volumetric, W^{vol} contributions:

$$W(\mathbf{F}) = W^{iso}(\mathbf{F}) + W^{vol}(\mathbf{F}). \tag{2}$$

3.1 Requirements for Hyperelastic Constitutive Laws

In the following, we briefly overview the main requirements that hyperelastic constitutive laws have to satisfy according to continuum mechanics theory. For more details, see [45, 46].

Thermodynamic Consistency: A hyperelastic material is one for which we postulate the existence of an elastic strain energy density function

$$W : \mathcal{GL}^+(3) \to \mathbb{R}, \quad \mathbf{F} \mapsto W(\mathbf{F}),$$

where $\mathcal{GL}^+(3)$ is the set of invertible second-order tensors with positive determinant. The laws of thermodynamics lead to the Clausius-Duhem inequality

$$\mathbf{P} : \dot{\mathbf{F}}^T - \dot{W} \geq 0,$$

where : denotes the tensor dot product. The Clausius-Duhem inequality is satisfied as an equality for an arbitrary $\dot{\mathbf{F}}$ by the following definition of the first Piola-Kirchhoff stress tensor

$$\mathbf{P} = \frac{\partial W}{\partial \mathbf{F}}. \tag{3}$$

Symmetry of the Stress Tensor: Balance of angular momentum implies the symmetry of the Cauchy stress tensor $\boldsymbol{\sigma} = J^{-1}\mathbf{P}\,\mathbf{F}^T$, and results in the fact that the strain energy density function $W(\mathbf{F})$ needs to satisfy

$$\mathbf{P}\,\mathbf{F}^T = \mathbf{F}\,\mathbf{P}^T \rightarrow \frac{\partial W}{\partial \mathbf{F}}\,\mathbf{F}^T = \mathbf{F}\,\frac{\partial W}{\partial \mathbf{F}}^T$$

Objectivity (Frame Indifference): Objectivity means independence from the choice of the observer, which in hyperelasticity can be expressed as

$$W(\mathbf{Q}\cdot\mathbf{F}) = W(\mathbf{F}) \quad \forall \mathbf{F} \in \mathcal{GL}^+(3), \quad \mathbf{Q} \in \mathcal{SO}(3),$$

with $\mathcal{SO}(3)$ as the 3D rotation group. It is straightforward to show that objectivity is automatically satisfied if W depends on $\mathbf{F}$ through $\mathbf{C}$, i.e. if $W = \tilde{W}(\mathbf{C})$.

Material Symmetry: The strain energy density function should reflect the desired type of material symmetry, e.g. isotropy or a specific class of anisotropy, which can be formalized as follows

$$W(\mathbf{Q}\cdot\mathbf{F}) = W(\mathbf{F}) \quad \forall \mathbf{F} \in \mathcal{GL}^+(3), \quad \mathbf{Q} \in G \subseteq \mathcal{O}(3),$$

where G is the symmetry group of the material.

Polyconvexity: Polyconvexity of the strain energy density function is a sufficient condition for the existence of solutions to boundary value problems with hyperelastic material behavior under general boundary conditions and body forces [47, 48]. $W(\mathbf{F})$ is polyconvex if and only if there exists a function $\mathcal{P}$, *convex* in its arguments, such that

$$W(\mathbf{F}) = \mathcal{P}(\mathbf{F}, \operatorname{adj}\mathbf{F}, \det\mathbf{F})$$

where $\operatorname{adj}\mathbf{F} = \det\mathbf{F}\,\mathbf{F}^{-\mathrm{T}}$.

Normalization of Stress and Strain Energy Density: In an undeformed configuration, i.e. for $\mathbf{F} = \mathbf{I}$, it must be

$$W(\mathbf{F} = \mathbf{I}) = 0 \quad \text{and} \quad \mathbf{P}(\mathbf{F} = \mathbf{I}) = \mathbf{0}.$$

In other words, an undeformed configuration implies no stresses and stores no energy.

Growth Condition: The growth condition requires that the strain energy density grows to infinity as the volumetric deformation tends to zero or to infinity, as follows

$$W(\mathbf{F}) \rightarrow \infty \quad \text{as } J \rightarrow 0^+ \text{ or } J \rightarrow \infty \quad \forall\, \mathbf{F} \in \mathcal{GL}^+(3).$$

Its physical meaning is that an infinitesimal material volume cannot grow to infinity or be compressed to a point.

Non-Negativity of the Strain Energy Density: The strain energy density also needs to satisfy

$$W(\mathbf{F}) \geq 0,$$

as a negative strain energy density is physically meaningless.

3.2 Enforcement of Constraints

Now that the requirements for elastic strain energy densities have been defined, we discuss how these constraints can be translated to language semantics. The grammar $\mathcal{G}_{NH}$ introduced in Section 2 leads to expressions that do not necessarily satisfy the constraints in Section 3.1. If we consider e.g. $W = (J - 0.5)^2 + 3 \cdot I_1^2$ (one of the examples in Figure 4 with $a = 0.5$), it is evident that this expression does not satisfy the normalization condition. In this section, we propose a process to embed some of the constitutive law constraints directly in the definition of the grammar and to apply additional semantic checks to generated expressions in order to ensure that the remaining constraints are satisfied. Therefore, we propose creating a grammar, and the corresponding language, such that its derived expressions comprise automatically valid strain energy density functions.

Intrinsic Constraints: Some of the requirements in Section 3.1 can be easily fulfilled a priori in the grammar construction. Thermodynamic consistency for hyperelastic materials is trivially satisfied; symmetry of the stress tensor and frame indifference are automatically fulfilled by considering W as a function of the deformation gradient through the right Cauchy-Green deformation tensor $\mathbf{C}$. In terms of material symmetry, from now on we focus on isotropic hyperelasticity; isotropy is automatically guaranteed by considering W as a function of the invariants of $\mathbf{C}$.

To satisfy normalization, we correct $W(\mathbf{F})$ as follows:

$$\tilde{W}(\mathbf{F}) = W(\mathbf{F}) + W^0 + \mathbf{P}^c : (\mathbf{F} - \mathbf{I}) \tag{4}$$

where W^0 and $\mathbf{P}^c$ are corrections to satisfy the strain energy density and the stress normalization, respectively, given by

$$\tilde{W}(\mathbf{F} = \mathbf{I}) = \mathbf{0} \rightarrow W^0 = -W|_{\mathbf{F}=\mathbf{I}}, \tag{5}$$

$$\tilde{\mathbf{P}}(\mathbf{F} = \mathbf{I}) = \frac{\partial \tilde{W}(\mathbf{F})}{\partial \mathbf{F}}\Big|_{\mathbf{F}=\mathbf{I}} = \mathbf{0} \rightarrow \mathbf{P}^c = -\frac{\partial W(\mathbf{F})}{\partial \mathbf{F}}\Big|_{\mathbf{F}=\mathbf{I}}. \tag{6}$$

Extrinsic Constraints: The remaining requirements in Section 3.1 are not satisfied a priori, but verified on the final produced expression. If they are not fulfilled, the expression is discarded. In order to empirically check if the growth condition holds, we choose $\mathbf{F}$ as the diagonal matrix:

$$\mathbf{F} = \begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

for which $J = a$. We set a to a large positive real number a_l to check the case $J \rightarrow \infty$ and to a small positive real number a_s for the case $J \rightarrow 0^+$. Then, we choose a large positive real number H and consider the growth condition satisfied if $W > H$ for $a = a_l$ and $a = a_s$. In our later numerical examples, we choose $a_l = 10^4$, $a_s = 10^{-4}$ and $H = 10^3$.

To check (again empirically) the non-negativity and the monotonicity of the strain energy density function, we consider one-parametric deformation gradients corresponding to several simple tests, namely Uni-axial Tension (UT), Bi-axial Tension (BT), Uni-axial Compression (UC), Bi-axial Compression (BC), Simple Shear (SS), and Pure Shear (PS) [2], as follows

$$\begin{aligned} &\mathbf{F}_{UT} = \begin{bmatrix} 1+\gamma & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{F}_{UC} = \begin{bmatrix} 1/(1+\gamma) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{F}_{SS} = \begin{bmatrix} 1 & \gamma & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ &\mathbf{F}_{BT} = \begin{bmatrix} 1+\gamma & 0 & 0 \\ 0 & 1+\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{F}_{BC} = \begin{bmatrix} 1/(1+\gamma) & 0 & 0 \\ 0 & 1/(1+\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{F}_{PS} = \begin{bmatrix} 1+\gamma & 0 & 0 \\ 0 & 1/(1+\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned} \tag{7}$$

with γ as a real non-negative parameter. $W(\mathbf{F}(\gamma))$ should be a positive and monotonically increasing function of γ. We then consider parameters γ_1, γ_2 with $\gamma_2 > \gamma_1$ and check that

$$0 < W(\mathbf{F}(\gamma_1)) < W(\mathbf{F}(\gamma_2)), \quad \forall \gamma_1, \gamma_2 \in (0, \infty), \tag{8}$$

If Equation 8 does not hold for a pair (γ_1, γ_2) for any of the loading conditions in 7, the mathematical expression is rejected.

Polyconvexity: The requisite of polyconvexity deserves a separate discussion. In view of the equivalence mentioned in Section 3.1 and of an additivity property [48], a possible simple choice of a polyconvex strain energy density is the following:

$$W(\mathbf{F}) = W_1(\det \mathbf{F}) + \mathrm{W}_2(\tilde{\mathrm{I}}_1(\mathbf{F}, \mathrm{det}\mathbf{F})) + \mathrm{W}_3(\tilde{\mathrm{I}}_2^{3/2}(\mathrm{adj}\mathbf{F}, \mathrm{det}\mathbf{F})),$$

with W_1, W_2 and W_3 convex and monotonically increasing with respect to their arguments. Since $\tilde{I}_1$ and $\tilde{I}_2^{3/2}$ are polyconvex [48], the resulting strain energy density function is naturally polyconvex. Note that this choice is quite restrictive, as we are not including any terms containing both $\tilde{I}_1$ and $\tilde{I}_2$ (since these mixed terms are not polyconvex [48]).

The approach we follow in this paper allows for more flexibility. As will become clear later, we work with a strain energy density function of the form

$$W = \hat{W}(J, \tilde{I}_1, \tilde{I}_2^{3/2}),$$

thus, we bias the grammar to derive polyconvex expressions through the choice of the exponent for $\tilde{I}_2$, however we do not strictly guarantee $\hat{W}$ convexity or monotonicity. Over polyconvexity, which may be regarded as an even too strong requirement for practical purposes, we prefer a more flexible grammar construction to allow for more exotic expressions to be generated. However, already at this point we would like to stress that the grammar construction fully controls the properties of the derived expressions; using mathematical tools from formal languages, these properties can be provably controlled in a rigorous manner [44], hence, in principle they provide the possibility to strictly enforce polyconvexity if desired.

Example. To exemplify the process of hyperelastic model generation including constraint enforcement, we refer back to $\mathcal{G}_{\text{NH}}$ and apply a minor modification. Without loss of generality, we consider $a = 0.5$, $b = 1.5$ to define a specific material and the isochoric invariant $\tilde{I}_1$ in place of I_1. We present the grammar construction in two ways. First, we rewrite $\mathcal{G}_{\text{NH}}$ upon implementation of the above modifications:

$$\begin{aligned}
&S = \{\ C\ \},\\
&\Phi = \{\ C,\ \Psi, L\ \},\\
&\Sigma = \{\ +, -, \cdot, ()^2, \tilde{I}_1, J,\ 0.5\ , 1, 1.5, 3\ \},\\
&R = \{\ C \rightarrow\ \Psi + \Psi, (\mathbf{1})\\
&\qquad \Psi \rightarrow L \cdot \Psi, (\mathbf{2})\\
&\qquad \Psi \rightarrow\ (L - L), (\mathbf{3})\\
&\qquad \Psi \rightarrow\ (\Psi)^2, (\mathbf{6})\\
&\qquad \Psi \rightarrow\ L, (\mathbf{7})\\
&\qquad L \rightarrow 0.5, (\mathbf{8})\\
&\qquad L \rightarrow 1, (\mathbf{9})\\
&\qquad L \rightarrow 1.5, (\mathbf{10})\\
&\qquad L \rightarrow 3, (\mathbf{11})\\
&\qquad L \rightarrow \tilde{I}_1, (\mathbf{12})\\
&\qquad L \rightarrow J(\mathbf{13})\ \}.
\end{aligned}$$

As we already did in Section 2.1, we generate an expression by consecutively applying a sequence of production rules, i.e. $r = [(\mathbf{1}), (\mathbf{2}), (\mathbf{2}), (\mathbf{6}), (\mathbf{3}), (\mathbf{9}), (\mathbf{6}), (\mathbf{13}), (\mathbf{8}), (\mathbf{11}), (\mathbf{2}), (\mathbf{9}), (\mathbf{7}), (\mathbf{12})]$, in a top-to-bottom-left-to-right fashion, as follows:

$$
\begin{aligned}
C &\xrightarrow{(1)} \Psi + \Psi, \\
&\xrightarrow{(2)} L \cdot \Psi + \Psi, \\
&\xrightarrow{(2)} L \cdot \Psi + L \cdot \Psi, \\
&\xrightarrow{(6)} L \cdot (\Psi)^2 + L \cdot \Psi, \\
&\xrightarrow{(3)} L \cdot (L - L)^2 + L \cdot \Psi, \\
&\xrightarrow{(9)} 1 \cdot (L - L)^2 + L \cdot \Psi, \\
&\xrightarrow{(6)} 1 \cdot (L - L)^2 + L \cdot (\Psi)^2, \\
&\xrightarrow{(13)} 1 \cdot (J - L)^2 + L \cdot (\Psi)^2, \\
&\xrightarrow{(8)} 1 \cdot (J - 0.5)^2 + L \cdot (\Psi)^2, \\
&\xrightarrow{(11)} 1 \cdot (J - 0.5)^2 + 3 \cdot (\Psi)^2, \\
&\xrightarrow{(2)} 1 \cdot (J - 0.5)^2 + 3 \cdot (C \cdot \Psi)^2, \\
&\xrightarrow{(9)} 1 \cdot (J - 0.5)^2 + 3 \cdot (1 \cdot \Psi)^2, \\
&\xrightarrow{(7)} 1 \cdot (J - 0.5)^2 + 3 \cdot (1 \cdot L)^2, \\
&\xrightarrow{(12)} 1 \cdot (J - 0.5)^2 + 3 \cdot (1 \cdot \tilde{I}_1)^2
\end{aligned}
$$

and derive $W(\mathbf{F}) = (J - 0.5)^2 + 3 \cdot \tilde{I}_1^2$. As a second alternative, we introduce two new non-terminal symbols Ψ^{iso} and Ψ^{vol} to indicate the isochoric and volumetric parts of the strain energy density, and define production rules for each of these non-terminals:

$$
\begin{aligned}
S &= \{\ C\ \}, \\
\Phi &= \{\ C,\ \Psi^{vol},\ \Psi^{iso}, L, L^{vol}, L^{iso}\ \}, \\
\Sigma &= \{\ +, -, \cdot, ()^2, \tilde{I}_1, J,\ 0.5\ , 1, 1.5, 3\ \}, \\
R &= \{\ C \rightarrow\ \Psi^{iso} + \Psi^{vol}, (\mathbf{1}) \\
&\qquad \Psi^{vol} \rightarrow L \cdot \Psi^{vol}, (\mathbf{2}) \\
&\qquad \Psi^{vol} \rightarrow\ (L^{vol} - L^{vol}), (\mathbf{3}) \\
&\qquad \Psi^{vol} \rightarrow\ (\Psi^{vol})^2, (\mathbf{4}) \\
&\qquad \Psi^{vol} \rightarrow\ L^{vol}, (\mathbf{5}) \\
&\qquad \Psi^{iso} \rightarrow L \cdot \Psi^{iso}, (\mathbf{6}) \\
&\qquad \Psi^{iso} \rightarrow\ (L^{iso} - L^{iso}), (\mathbf{7}) \\
&\qquad \Psi^{iso} \rightarrow\ (\Psi^{iso})^2, (\mathbf{8}) \\
&\qquad \Psi^{iso} \rightarrow\ L^{iso}, (\mathbf{9}) \\
&\qquad L^{vol} \rightarrow 0.5, (\mathbf{10}) \\
&\qquad L \rightarrow 1, (\mathbf{11}) \\
&\qquad L \rightarrow 1.5, (\mathbf{12}) \\
&\qquad L^{iso} \rightarrow 3, (\mathbf{13}) \\
&\qquad L^{iso} \rightarrow \tilde{I}_1, (\mathbf{14}) \\
&\qquad L^{vol} \rightarrow J (\mathbf{15})\ \},
\end{aligned}
$$

With this grammar definition, it is possible to derive expressions which distinguish between volumetric and deviatoric (isochoric) contributions, which is a common choice in constitutive modeling of hy-

perelastic materials. From this grammar we again sample a sequence of production rules, i.e. $r = [(\mathbf{1}), (\mathbf{2}), (\mathbf{6}), (\mathbf{4}), (\mathbf{3}), (\mathbf{11}), (\mathbf{8}), (\mathbf{15}), (\mathbf{10}), (\mathbf{13}), (\mathbf{6}), (\mathbf{11}), (\mathbf{9}), (\mathbf{14})]$, and consecutively apply them in a top-to-bottom-left-to-right fashion to re-write the non-terminals until we derive a terminal expression, as follows:

$$
\begin{aligned}
C &\xrightarrow{(\mathbf{1})} \Psi^{vol} + \Psi^{iso}, \\
&\xrightarrow{(\mathbf{2})} L \cdot \Psi^{vol} + \Psi^{iso}, \\
&\xrightarrow{(\mathbf{6})} L \cdot \Psi^{vol} + L \cdot \Psi^{iso}, \\
&\xrightarrow{(\mathbf{4}} L \cdot (\Psi^{vol})^2 + L \cdot \Psi^{iso}, \\
&\xrightarrow{(\mathbf{3})} L \cdot (L^{vol} - L^{vol})^2 + L \cdot \Psi^{iso}, \\
&\xrightarrow{(\mathbf{11})} 1 \cdot (L^{vol} - L^{vol})^2 + L \cdot \Psi^{iso}, \\
&\xrightarrow{(\mathbf{8})} 1 \cdot (L^{vol} - L^{vol})^2 + L \cdot (\Psi^{iso})^2, \\
&\xrightarrow{(\mathbf{15})} 1 \cdot (J - L^{vol})^2 + L \cdot (\Psi^{iso})^2, \\
&\xrightarrow{(\mathbf{10})} 1 \cdot (J - 0.5)^2 + L \cdot (\Psi^{iso})^2, \\
&\xrightarrow{(\mathbf{13})} 1 \cdot (J - 0.5)^2 + 3 \cdot (\Psi^{iso})^2, \\
&\xrightarrow{(\mathbf{6})} 1 \cdot (J - 0.5)^2 + 3 \cdot (C \cdot \Psi^{iso})^2, \\
&\xrightarrow{(\mathbf{11})} 1 \cdot (J - 0.5)^2 + 3 \cdot (1 \cdot \Psi^{iso})^2, \\
&\xrightarrow{(\mathbf{9})} 1 \cdot (J - 0.5)^2 + 3 \cdot (1 \cdot L)^2, \\
&\xrightarrow{(\mathbf{14})} 1 \cdot (J - 0.5)^2 + 3 \cdot (1 \cdot \tilde{I}_1)^2
\end{aligned}
$$

obtaining again $W(\mathbf{F}) = (J - 0.5)^2 + 3 \cdot \tilde{I}_1^2$. Note that with this second version, where separate non-terminals are considered for the isochoric and volumetric parts of the strain energy density, the number of grammar rules is larger since we need rules for each non-terminal. For the sake of simplicity, in the following developments of this paper we will consider a single non-terminal Ψ and not distinguish between volumetric and deviatoric contributions. However, considering both terms separately would be straightforward as shown above.

The obtained expression for $W(\mathbf{F})$ does not satisfy the normalization conditions, because $W(\mathbf{F} = \mathbf{I}) = 27.25$ and $\mathbf{P}(\mathbf{F} = \mathbf{I}) = \mathbf{I}$. Thus, we perform the corrections (5) and (6) and derive the expression:

$$\tilde{W}(\mathbf{F}) = 3 \cdot \tilde{I}_1^2 + (J - 0.5)^2 - 27.25 + \mathbf{I} : (\mathbf{F} - \mathbf{I}). \tag{9}$$

Performing the corrections is equivalent to adding a sub-tree to the original expression, see Figure 5. In Figure 5 we also provide the predicted displacement field for the benchmark in [2], reproduced in Figure 6, considering $\tilde{W}(\mathbf{F})$ from eq. (9).

We then execute the remaining empirical checks on volumetric growth and non-negativity, both of which hold in this case. Therefore, we conclude that the derived expression can indeed be deemed a constitutive law that can characterize a hyperelastic material.

3.3 Generating the Language of Hyperelastic Materials

So far we considered a simple grammar, based on which we could generate simple expressions which we guaranteed to be valid (simple) constitutive laws. In order to obtain more flexible expressions, we now consider a more complicated alphabet of terms, include production rules with more operations, and derive constants via recursive substitution of non-terminals instead of explicitly defining production rules that re-write non-terminals with reals. In the choice of the primitives we consider logarithmic, exponential, monomials and other operations that often appear in constitutive relations. To keep a reasonably compact grammar, we do not consider different non-terminals and production rules for the isochoric and volumetric parts of the strain energy density, but one non-terminal for both. However, if desired, this can be changed as exemplified earlier. Finally, we define the energy and stress corrections as production rules for the non-terminal Ψ. The resulting grammar of hyperelastic materials, $\mathcal{G}_{\text{HM}}$, is then defined as follows:

$$S = \{C\},$$

$$
\begin{aligned}
&\Phi = \{C, \Psi, \Psi^0, P^c, L, M, P, D\},\\
&\Sigma = \{+, -, /, \cdot, ()^2, ()^3, -\log, \exp, \tilde{I}_1, \tilde{I}_2, J, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \frac{\partial()}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), |_{\mathbf{F}=\mathbf{I}}\},\\
&R = \{C \to \Psi + \Psi^0 + P^c, (\mathbf{1})\\
&\qquad \Psi \to \Psi + \Psi, (\mathbf{2})\\
&\qquad \Psi^0 \to -\Psi|_{\mathbf{F}=\mathbf{I}}, (\mathbf{3})\\
&\qquad P^c \to -\frac{\partial \Psi}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), (\mathbf{4})\\
&\qquad \Psi \to L, (\mathbf{5})\\
&\qquad L \to \ (L+L)\,(\mathbf{6})\\
&\qquad L \to \ (L-L)\,(\mathbf{7})\\
&\qquad L \to \ (L/R)\,(\mathbf{8})\\
&\qquad L \to \ (R \cdot L)\,(\mathbf{9})\\
&\qquad L \to \ (L)^2\,, (\mathbf{10})\\
&\qquad L \to \ (L)^3\,, (\mathbf{11})\\
&\qquad L \to \ -\log(L)\,, (\mathbf{12})\\
&\qquad L \to \ \exp(L)\,(\mathbf{13})\\
&\qquad L \to \ \tilde{I}_1\,(\mathbf{14})\\
&\qquad L \to \ \tilde{I}_2^{3/2}\,(\mathbf{15})\\
&\qquad L \to \ J\,(\mathbf{16})\\
&\qquad L \to \ M, (\mathbf{17})\\
&\qquad M \to \ P.P, (\mathbf{18})\\
&\qquad P \to \ D\,, (\mathbf{19})\\
&\qquad P \to \ DP, (\mathbf{20})\\
&\qquad D \to \ 0\,, (\mathbf{21})\\
&\qquad D \to \ 1\,, (\mathbf{22})\\
&\qquad D \to \ 2\,, (\mathbf{23})\\
&\qquad D \to \ 3\,, (\mathbf{24})\\
&\qquad D \to \ 4\,, (\mathbf{25})\\
&\qquad D \to \ 5\,, (\mathbf{26})\\
&\qquad D \to \ 6\,(\mathbf{27}),\\
&\qquad D \to \ 7\,, (\mathbf{28})\\
&\qquad D \to \ 8\,, (\mathbf{29})\\
&\qquad D \to \ 9\,(\mathbf{30})\}
\end{aligned}
$$

In $\mathcal{G}_{\text{HM}}$, we purposely define multiple production rules for the non-terminals L, P, D; this choice enforces recursions that increase the expressivity of the grammar. When two rules have the same left-hand side, we randomly select one to apply. In CFGs non-terminals should be treated independently in each production rule; however, for the purpose of the normalization correction (see eq. (4)) we enforce that the same non-terminal Ψ is substituted to the left-hand side in the production rules $(\mathbf{1}), (\mathbf{3})$, and $(\mathbf{4})$. Now we return to eq. (9), and show how this simple constitutive law can be derived also from the present more complex grammar via the sequence of production rules $r = [(\mathbf{1}), (\mathbf{3}), (\mathbf{4}), (\mathbf{2}), (\mathbf{5}), (\mathbf{10}), (\mathbf{7}), (\mathbf{16}), (\mathbf{17}), (\mathbf{18}), (\mathbf{19}), (\mathbf{21}), (\mathbf{26}), (\mathbf{5}), (\mathbf{9}), (\mathbf{18}), (\mathbf{21}), (\mathbf{10}), (\mathbf{14})]$ with top-

to-bottom-left-to-right ordering:

$$
\begin{aligned}
C &\xrightarrow{(1)} \Psi + \Psi^0 + P^c, \\
&\xrightarrow{(3)} \Psi - (\Psi)|_{\mathbf{F}=\mathbf{I}} + P^c, \\
&\xrightarrow{(4)} \Psi - (\Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial \Psi}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(2)} \Psi + \Psi - (\Psi + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial (\Psi + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(5)} L + \Psi - (L + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial (L + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(10)} (L)^2 + \Psi - ((L)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial (L + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(7)} (L-L)^2 + \Psi - ((L-L)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((L-L) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(16)} (J-L)^2 + \Psi - ((J-L)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-L) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(17)} (J-M)^2 + \Psi - ((J-M)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-M) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(18)} (J-P.P)^2 + \Psi - ((J-P.P)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-P.P) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(19)} (J-D.D)^2 + \Psi - ((J-D.D)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-D.D) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(21)} (J-0.D)^2 + \Psi - ((J-0.D)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.D) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(26)} (J-0.5)^2 + \Psi - ((J-0.5)^2 + \Psi)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5) + \Psi)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(5)} (J-0.5)^2 + L - ((J-0.5)^2 + L)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5) + L)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(9)} (J-0.5)^2 + M \cdot L - ((J-0.5) + M \cdot L)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5)^2 + M \cdot L)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(18)} (J-0.5)^2 + P.P \cdot L - ((J-0.5) + P.P \cdot L)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5)^2 + P.P \cdot L)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(24)} (J-0.5)^2 + 3.P \cdot L - ((J-0.5) + 3.P \cdot L)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5)^2 + 3.P \cdot L)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(21)} (J-0.5)^2 + 3.0 \cdot L - ((J-0.5) + 3.0 \cdot L)|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5)^2 + 3.0 \cdot L)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(10)} (J-0.5)^2 + 3.0 \cdot (L)^2 - ((J-0.5) + 3.0 \cdot (L)^2))|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5)^2 + 3.0 \cdot (L)^2)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), \\
&\xrightarrow{(14)} (J-0.5)^2 + 3.0 \cdot (I_1)^2 - ((J-0.5) + 3.0 \cdot (I_1)^2))|_{\mathbf{F}=\mathbf{I}} - \frac{\partial ((J-0.5)^2 + 3.0 \cdot (I_1)^2)}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}).
\end{aligned}
$$

By evaluating the production rules $r = [(\mathbf{3}), (\mathbf{4})]$ we derive eq. (9). It takes more steps to complete the derivation and obtain the strain energy density expression because $\mathcal{G}_{\text{HM}}$ contains more numerous and more general production rules than $\mathcal{G}_{\text{NH}}$. As usual, the growth, non-negativity and monotonicity conditions are empirically checked a posteriori.

To summarize, in the grammar of hyperelastic materials $\mathcal{G}_{\text{HM}}$ the thermodynamic consistency, the symmetry of the stress tensor, the objectivity and material symmetry conditions are satisfied a priori, the normalization of the stress and the strain energy density are satisfied by construction and the growth, non-negativity and monotonicity conditions are empirically assessed a posteriori by testing the obtained expression on a number of simple one-parametric deformation paths.

The proposed approach is highly versatile, because with minimal changes to the grammar one can describe different types of materials. For example in the case where anisotropy is present due to fiber reinforcement, additional invariants need to be considered to describe the deformation along the direction of anisotropy. For example in a material reinforced

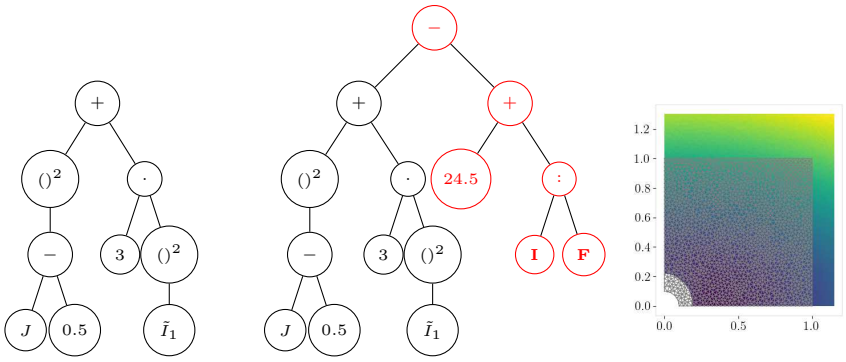


Figure 5: Left: The tree representation of $W = 3 \cdot \tilde{I}_1^2 + (J - 0.5)^2$. Middle: The tree representation of the expression $\tilde{W} = 3 \cdot \tilde{I}_1^2 + (J - 0.5)^2 - 27.25 + \mathbf{I} : (\mathbf{F} - \mathbf{I})$, with the correction as a sub-tree attached to the original expression. Right: The displacement plot with $\tilde{W}$ used as constitutive law for the boundary value problem in Figure 6 with $\delta = 0.3$.

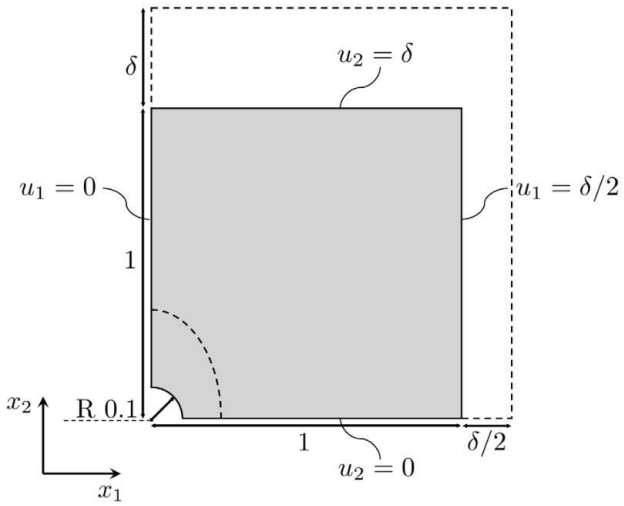


Figure 6: Benchmark boundary value problem considered in this work, adapted from [2].

with two fibers, $\mathcal{G}_{\text{HM}}$ can be further adjusted by considering the production rules [49]:

$$
\begin{aligned}
L &\rightarrow J_4, \\
L &\rightarrow J_5, \\
L &\rightarrow J_6, \\
L &\rightarrow J_7,
\end{aligned}
$$

and adding J_4, J_5, J_6, J_7, i.e. the anisotropic invariants for the two fiber families, to the alphabet. The ensuing language $\mathcal{L}(\mathcal{G}_{\text{HM}})$ will include constitutive models for this anisotropy case. Clearly, analogous modifications can be applied for more complex anisotropy cases.

We provided the derivation of one expression by choosing a sequence of production rules from $\mathcal{G}_{\text{HM}}$. All the possible derivation trees that the grammar can produce comprise the Language of Hyperelastic Materials $\mathcal{L}(\mathcal{G}_{\text{HM}})$. In the next section, we elaborate on how this language can be used for the automated generation of a library of constitutive models, see the first block of Figure 1.

3.4 Deriving Parsimonious Expressions

The recursive substitution process in the grammar of hyperelastic materials $\mathcal{G}_{\text{HM}}$ is not guaranteed to stop nor to provide a parsimonious expression, i.e. a shallow tree. To alleviate theses issues, the tree generation process can be manipulated so as to enforce or favor the generation of shallow trees. There are two main options. One option is to attribute different

probabilities to the production rules that share the same left-hand side, favoring those that produce terminal strings. Another alternative is to specify a fixed maximum number of operations to obtain the final expression. In this subsection, we briefly discuss these two alternative procedures.

Probabilistic Context-Free Grammars: As discussed earlier, in a CFG several production rules may have the same non-terminal on their left-hand side; as a result, a criterion is needed to select one of these production rules during the tree generation. A naive approach would consider a uniform probability over all production rules for a specific non-terminal. A greater flexibility and efficiency can be obtained using Probabilistic Context-Free Grammars (PCFGs) [50, 51, 52, 53]. PCFGs can be constructed from CFGs by assigning probabilities to the grammar production rules with the same non-terminal on the left-hand side, with the sum of the probabilities summing up to one. The probability for each rule can be assigned by the user or learned from the data. As an example, a PCFG can be obtained from $\mathcal{G}_{\text{NH}}$ (for $a = 0.5, b = 1.5$) by assigning probabilities to its production rules (given in square brackets) as follows

$$
\begin{aligned}
&S = \{\ C\ \},\\
&\Phi = \{\ C,\ \Psi, L\ \},\\
&\Sigma = \{\ +, -, \cdot, (\cdot)^2, \tilde{I}_1, J,\ 0.5\ , 1, 1.5, 3\ \},\\
&R = \{\ C \rightarrow [1]\ \ \Psi + \Psi, (\mathbf{1})\\
&\qquad\ \ \Psi \rightarrow\ [0.3]\ C \cdot \Psi, (\mathbf{2})\\
&\qquad\ \ \Psi \rightarrow\ [0.1]\ (C - C), (\mathbf{3})\\
&\qquad\ \ \Psi \rightarrow\ [0.1]\ (\Psi)^2, (\mathbf{4})\\
&\qquad\ \ \Psi \rightarrow\ [0.5]\ L, (\mathbf{5})\\
&\qquad\ \ L \rightarrow [0.1]\ 0.5, (\mathbf{6})\\
&\qquad\ \ L \rightarrow [0.1]\ 1, (\mathbf{7})\\
&\qquad\ \ L \rightarrow [0.1]\ 1.5, (\mathbf{8})\\
&\qquad\ \ L \rightarrow [0.1]\ 3, (\mathbf{9})\\
&\qquad\ \ L \rightarrow [0.3]\ \tilde{I}_1, (\mathbf{10})\\
&\qquad\ \ L \rightarrow [0.3]\ J(\mathbf{11})\ \}.
\end{aligned}
$$

Clearly, the probability of a constitutive law is given by the product of all probabilities associated with the rules chosen for its derivation, i.e.

$$P(W) = \prod_{i=1}^{N_p} P(r_i), \tag{10}$$

where $r = [r_1, ..., r_{N_p}]$ is the sequence of production rules that derive W. It is also clear that the probabilities of all possible final expressions sum up to one [52]. For deriving a deeper tree, i.e. a more complex expression, a longer sequence r is required, which includes a larger number of production rules and is thus associated to a lower probability. Even though this is an important property that can be used to favor the generation of parsimonious expressions, it requires a careful design of the grammar, as unrealistically low probabilities may be obtained already for simple models. For example, with $\mathcal{G}_{\text{NH}}$ the sequence of rules that produces the Neo-Hookean model $W = 0.5 \cdot (\tilde{I}_1 - 3) + 1.5 \cdot (J - 1)^2$ is $r = [(1), (2), (3), (5), (10), (9), (8), (11), (7)]$, and the probability of this model being produced from the grammar (with the individual rule probabilities in the previous example) is $P(W) = 1.35 \times 10^{-6}$.

Clearly, it is possible to modify the grammar and/or the individual rule probabilities to influence the probability of the resulting model form, but such modifications would require hand-engineering, thereby introducing bias in the model generation. Thus, PCFGs are not well suited for generation of expressions. Moreover, PCFG are not efficient for performing data-driven discovery tasks for long expressions because these will be discovered with a low probability. They are frequently used for tasks involving parsing of expressions, where the probability of each rule is learned from the data and not manually assigned by a user. For example, we could consider a probabilistic grammar if we were given a library of constitutive laws and wishes to determine the probabilities of individual rules being used in deriving a valid expression. This could be useful in a scenario where there is uncertainty about the number and form of the rules used to derive expressions. In such a case, we could define a more general grammar than we believe is needed and then, by learning suitable probabilities, we could discover the rules used in generating the library. Probabilistic grammars are also useful in understanding the decoding process that we employ in Section 4.

Constraining the Number of Operations in a Tree Derivation: The second alternative is to a priori decide the maximum number of operations that the grammar is allowed to perform in order to derive an expression. To realize

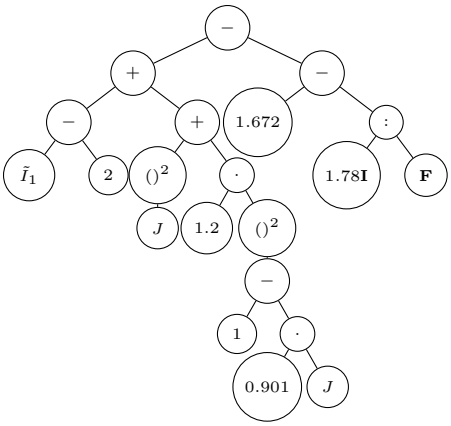
this constraint, we construct the grammar rules such that they do not allow for long recursions. For this purpose, we introduce a non-terminal symbol for each operation, e.g. by transforming rule $(\mathbf{2})$ from $\Psi \rightarrow \Psi + \Psi$ to $\Psi \rightarrow \Psi^1 + \Psi^2$, and for each of these non-terminal symbols we define unary and binary operations that in one step derive terminal nodes. In this way, we derive expressions of pre-set maximum length and complexity. We exemplify how this process works in practice in the next section.

3.5 Automatic Generation of an Hyperelastic Material Model Library

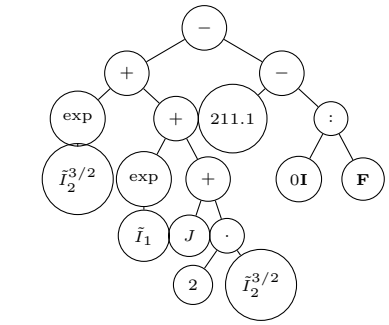
In this section, we demonstrate that the grammar-based generation of elastic strain energy density functions proposed in the previous sections can be used for the automatic creation of a material model library, which can possibly serve as the basis for sparse regression approaches such as those in [2, 27]. Moreover, we develop an automated computational pipeline which, once the mathematical expression for a valid constitutive law is generated, is used to produce the finite element code needed to solve boundary value problems where the material behavior obeys such a constitutive law. For the hyperelastic model library generation, we deploy a CFG for deriving mathematical expressions. As illustrated earlier, the grammar provides a systematic way of checking if the expressions are syntactically and semantically valid, and it is chosen to be as expressive as possible. For the purpose of obtaining parsimonious expressions, we no longer use the grammar $\mathcal{G}_{\text{HM}}$ defined in Section 3.3, but the following grammar, $\mathcal{G}_{\text{HMp}}$:

$$
\begin{aligned}
&S = \{\, C \,\},\\
&\Sigma = \{+, -, /, \cdot, (\cdot)^2, (\cdot)^3, -\log, \exp, \tilde{I}_1, \tilde{I}_2, J, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,\ \frac{\partial()}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}),,\ |_{\mathbf{F}=\mathbf{I}}\},\\
&R = \{C \rightarrow \Psi + \Psi^0 + P^c, (\mathbf{1})\\
&\qquad \Psi \rightarrow\ \Psi^1 + \Psi^2 + \Psi^3 + \Psi^4, (\mathbf{2})\\
&\qquad \Psi^0 \rightarrow -\Psi|_{\mathbf{F}=\mathbf{I}}, (\mathbf{3})\\
&\qquad \Psi^1 \rightarrow L\ (\mathbf{4})|\ U(\mathbf{5}) \mid Y(\mathbf{6}) \mid T(\mathbf{7})\ ,\\
&\qquad \Psi^2 \rightarrow L(\mathbf{8}) \mid U(\mathbf{9}) \mid Y(\mathbf{10}) \mid T(\mathbf{11})\ ,\\
&\qquad \Psi^3 \rightarrow L(\mathbf{12}) \mid U(\mathbf{13}) \mid Y(\mathbf{14}) \mid T(\mathbf{15})\ ,\\
&\qquad \Psi^4 \rightarrow L(\mathbf{16}) \mid U(\mathbf{17}) \mid Y(\mathbf{18}) \mid T(\mathbf{19})\ ,\\
&\qquad P^c \rightarrow -\frac{\partial \Psi}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F}-\mathbf{I}), (\mathbf{20})\\
&\qquad T \rightarrow (Y)^2(\mathbf{21})|(Y)^3(\mathbf{22})|\exp(Y)(\mathbf{23})| - \log(Y)(\mathbf{24}),\\
&\qquad Y \rightarrow\ (V+O)\ (\mathbf{25})|\ (V-O)(\mathbf{26}) \mid (V/O)(\mathbf{27}) \mid (V\cdot O)(\mathbf{28}) \mid (V)^2(\mathbf{29}) \mid (V)^3(\mathbf{30})\\
&\qquad U \rightarrow -\log(L)(\mathbf{31}) \mid \exp(L)(\mathbf{32})\\
&\qquad L \rightarrow V(\mathbf{33}) \mid O(\mathbf{34}),\\
&\qquad V \rightarrow \tilde{I}_1(\mathbf{35}) \mid \tilde{I}_2^{3/2}(\mathbf{36}) \mid J(\mathbf{37}),\\
&\qquad O \rightarrow P.P, (\mathbf{38})\\
&\qquad P \rightarrow D(\mathbf{39}) \mid DP(\mathbf{40}),\\
&\qquad D \rightarrow\ 0(\mathbf{41}) \mid 1(\mathbf{42}) \mid 2(\mathbf{43}) \mid 3(\mathbf{44}) \mid 4(\mathbf{45}) \mid 5(\mathbf{46}) \mid 6(\mathbf{47}) \mid 7(\mathbf{48}) \mid 8(\mathbf{49}) \mid 9(\mathbf{50})\ \},\\
&\Phi = \{\, C, \Psi, \Psi^0, \Psi^1, \Psi^2, \Psi^3, \Psi^4, P^c, T, L, U, Y, O, D, P\ \}
\end{aligned}
\tag{11}
$$

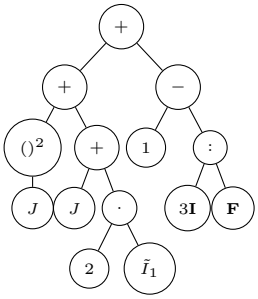
Here, U stands for unary, Y for binary, T for combinations of unary and binary operations, V for invariants, and L for both invariants and constants. We also denote with O real numbers, with D integers and with P parts of real numbers. We use the symbol "|" to separate rules with the same non-terminal on the left-hand side, for which we consider a uniform probability. As explained in Section 3.4, in this new grammar we limit the left-hand side recursions in the grammar production rules in order to create shallow trees that correspond to parsimonious expressions by introducing non-terminal symbols and operations of predetermined depth. For this reason, we define the grammar rule $(\mathbf{2})$ of $\mathcal{G}_{\text{HMp}}$ as $\Psi \rightarrow \Psi^1 + \Psi^2 + \Psi^3 + \Psi^4$ instead of $\Psi \rightarrow \Psi + \Psi$ and introduce the L, U, Y, T to derive terminal nodes in less steps. Therefore, we enforce that the strain energy density is represented by a tree of maximum depth equal to six. We use this grammar to produce syntactically valid expressions of elastic strain energy densities, which also satisfy intrinsic semantic constraints. The full semantic validity of these expressions is then assessed by checking if the growth and the non-negativity conditions hold, see Section 3.2. If these conditions hold, the expression is accepted as a valid constitutive law; if not, a new expression is constructed. Figure 7 illustrates several examples of valid final expressions.



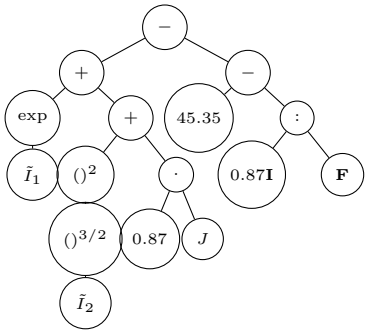
$2.0 \cdot I1 + J^2 + 1.23 \cdot (1.0 - 0.901 \cdot J)^2 - 1.672 - 1.78\mathbf{I} : \mathbf{F}$



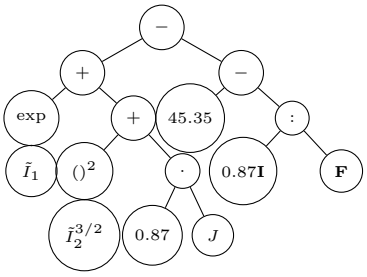
$2.0 \cdot \tilde{I}_2^{3/2} + \exp(\tilde{I}_1) + \exp(\tilde{I}_2^{3/2}) - 211.1 + 0\mathbf{I} : \mathbf{F}$



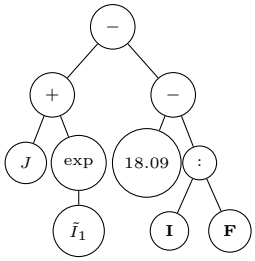
$2.0 \cdot I1 + J + J^2 + 1.0 - 3\mathbf{I} : \mathbf{F}$



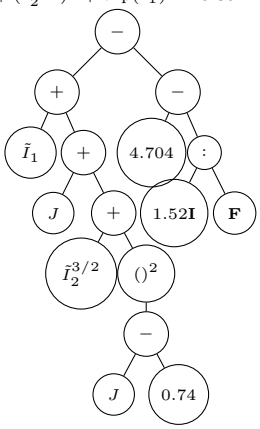
$0.87 \cdot J + (\tilde{I}_2^{3/2})^2 + \exp(\tilde{I}_1) - 45.35 - 0.87\mathbf{I} : \mathbf{F}$



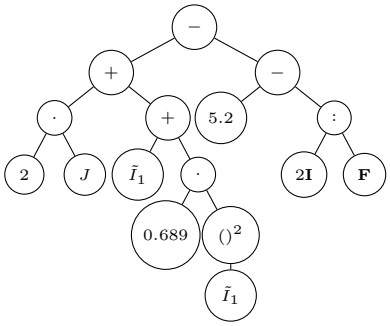
$(\tilde{I}_2^{3/2})^2 + 0.87 \cdot J + exp(\tilde{I}_1) - 45.35 - 0.871\mathbf{I} : \mathbf{F}$



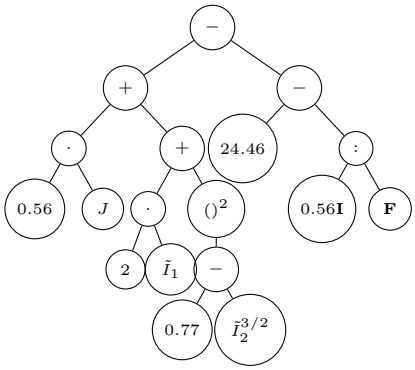
$J + \exp(\tilde{I}_1) - 18.09 - \mathbf{I} : \mathbf{F}$



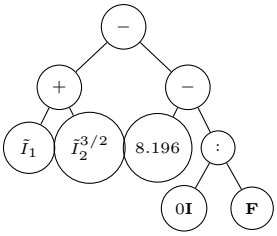
$\tilde{I}_1 + \tilde{I}_2^{3/2} + J + (J - 0.74)^2 - 4.704 - 1.52\mathbf{I} : \mathbf{F}$



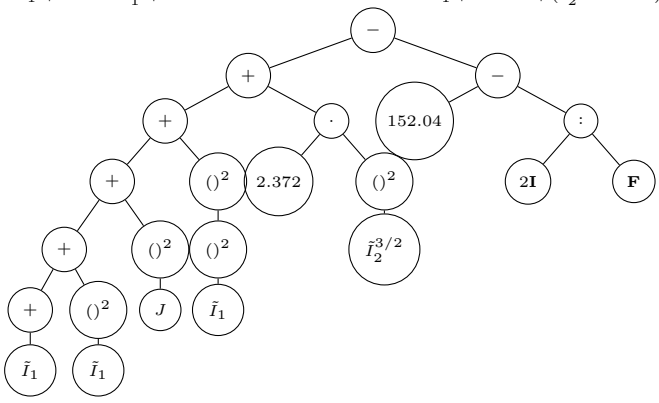
$\tilde{I}_1 + 0.689 \cdot \tilde{I}_1^2 + 2 \cdot J - 5.2 - 2\mathbf{I} : \mathbf{F}$



$2.0 \cdot \tilde{I}_1 + 0.56 \cdot J + (\tilde{I}_2^{3/2} - 0.77)^2 - 24.46 - 0.56\mathbf{I} : \mathbf{F}$

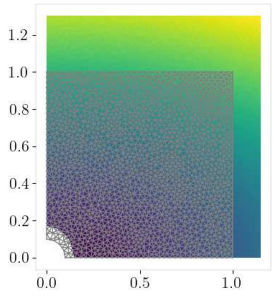


$\tilde{I}_1 + \tilde{I}_2^{3/2} - 8.196 - 0\mathbf{I} : \mathbf{F}$

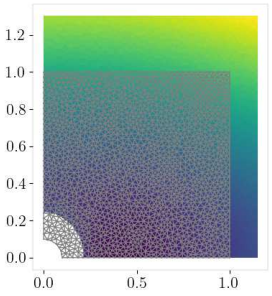


$I_1 + I_1^2 + J^2 + (I_1^2)^2 + 2.372 \cdot (I_2^{3/2})^2 - 152.044 - 3\mathbf{I} : \mathbf{F}$

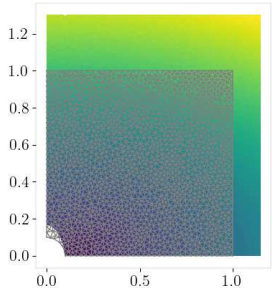
Figure 7: We present different constitutive laws from the language of hyperelastic materials $\mathcal{L}(\mathcal{G}_{\text{HMp}})$. These expressions satisfy all the constitutive law constraints.



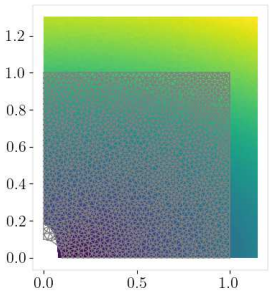
$2 \cdot \tilde{I}_1 + J + J^2 + 1.0 - 3\mathbf{I} : \mathbf{F}$



$2 \cdot \tilde{I}_1 + J^2 + 1.232 \cdot (1.0 - 0.901 \cdot J)^2 - 1.672 - 1.78\mathbf{I} : \mathbf{F}$



$\tilde{I}_1 + \tilde{I}_2^{3/2} + 2.509 \cdot \exp(\tilde{I}_1) + \exp(\tilde{I}_2^{3/2}) - 239.2$



$J + exp(\tilde{I}_1) - 18.09 - \mathbf{I} : \mathbf{F}$

Figure 8: We choose four constitutive law relations out of the ones presented in Figure 7 and perform a finite element simulation for the benchmark boundary value problem in Figure 6. We present the results for $\delta = 0.3$.

Clearly, the total number of models (or trees) that a library can potentially contain depends on the grammar and can be computed using combinatorial calculus. In this work, we do not generate all possible models stemming from $\hat{\mathcal{G}}_{\text{HM}}$, but fix upfront a desired number, and stop the generation process right after obtaining a number of valid models equal to the target. The automatic generation of the constitutive law library needs to be performed using symbolic computations, i.e. in SymPy [54]. The wall-clock time for the generation of a tree on a single processor is of order $O(10^{-4})$ seconds computed on an Alienware m16 Laptop with an Intel i9-13900HX CPU and 32GB RAM. In our experience, the probability of a generated tree to be accepted is about 50%; the entirety of the discarded trees violates the growth condition of the strain energy density function, whereas about 33% violate both the growth and the non-negativity conditions. To further speed up the process we leverage the independence of the generation process for different trees and use multi-threading, whereby each tree is generated by a different core of the CPU in an asynchronous manner. Thus, the total time for a model library generation depends on the number of available CPU units. For example the process of generating $1,000,000$ valid trees, with 100 CPU units available, would take a few minutes. This numbers naturally depend on the chosen number of operations in the grammar rule $(\mathbf{2})$, and we expect the acceptance rate of a tree to drop as the number increases, so the average time to produce a tree to increase.

Once a material model library is created automatically, for its practical use it is of fundamental importance to automate the process leading from availability of a constitutive law to solution of a boundary value problem embedding such law. The goal should be to seamlessly integrate the library creation with finite element simulations without having to perform changes to a finite element code for every different strain energy density function. This requires the possibility to use automatic differentiation to derive the stress and the tangent stiffness tensors from the elastic strain energy density [55], and this possibility is provided by modern finite element tools such as Fenics [56, 57, 58] or AceGEN/AceFEM [55]. In Figure 8, we present the displacement fields computed by solving the boundary value problem in Figure 6 with $\delta = 0.3$, for four of the hyperelastic constitutive models in Figure 7, with the Fenics library. We discretize the domain using 1024 quadratic triangular elements with a three-point Gauss quadrature rule.

The code for the generation of the hyperelastic material model library and the code for solving the boundary value problem in Figure 6 will be made publicly available at the time of publication.

4 Data-Driven Discovery of Hyperelastic Constitutive Laws

One of the possible motivations for generating the Language of Hyperelastic Materials is to use it within the context of data-driven constitutive model discovery. As discussed already in the introduction, constitutive model identification is typically formulated as an inverse problem, where the unknown parameters in a chosen model are to be inferred on the basis of available experimental information from a tested system. Constitutive model discovery goes one step further, as it integrates model selection with parameter identification [27]. In Section 1 we already discussed the properties that a symbolic regression method needs to possess in order to be useful for constitutive law discovery. In this paper, we have chosen to work with methods based on grammar due to their advantages: they overcome human bias, since they are not restricted to producing known models or combinations thereof, and the grammar construction imposes semantic and syntactic constraints to the constitutive law expressions. In this section, we aim at showcasing the solution of the inverse data-driven constitutive law discovery problem using a method that features reasonable complexity, accommodates different primitive operations without training instabilities, and generalizes across different materials. For this purpose, we consider a combination of the Recursive Tree Grammar Variational Autoencoder (RTGVAE) method proposed by Paassen et. al. [42], applied to the Language of Hyperelastic Materials formulated in the previous section, and the Covariant Matrix Adaptation Evolutionary Strategy [59] (CMA-ES) for gradient-free optimization. Our purpose in choosing RTGVAE is to show that the proposed methodology can be seamlessly integrated with an already established symbolic regression method. However, in principle any other symbolic regression method could be considered, e.g. a LSPT model, in combination with our grammar-generated hyperelastic constitutive laws. In symbolic regression methods that consider grammar, the latent space is constructed by encoding sequences of grammar rules. Therefore, when performing the data-drive discovery, the models produced in the process of optimization are biased to satisfies the grammar and thus be semantically and syntactically valid. If we considered a LSPT model, for example, the latent space would not be constructed by encoding grammar rules, which means that there would be a higher probability of obtaining invalid expressions during the discovery process.

4.1 Symbolic Regression Using Grammar and Model Discovery

We discuss in Sections 1 and 2 how trees can be characterized via grammar rules instead of their node labels and connectivity, see Appendix B for more information. In Appendix B we also discuss the operations of extracting a nested list of rules given a tree, called parsing, and deriving a tree given a list of rules, called generation. A *tree VAE* is built on these two operations. More specifically, its encoder extracts the nested list of production rules that derive different trees (inputs) and produces its low-dimensional vectorial encoding. The decoder takes an encoding and provides the most probable nested list of rules that corresponds to the encoding (outputs). The VAE is trained to provide, with high probability, the correct list of rules given a latent vector. In the next paragraphs, we discuss in more detail how the encoder, the decoder and the loss function are defined.

Tree Encoder: The purpose of a tree encoder is to provide a systematic way to represent trees that adhere to a certain set of grammar rules using a low-dimensional vectorial representation. This encoding represents hierarchical or nested lists as a vector, which is a data structure that is easier to handle in computer programs. We explain how the encoder works by comparing encoding to the bottom-up parsing process, as they are closely connected. In parallel to parsing, the encoder assigns a representation to each child and recursively maps the representations of the children to the representation of the parent using a function f^r for each rule r. In [42], the f^r functions are defined as Recursive Neural Networks [60, 61, 62] to capture the hierarchical structure of the trees. This process begins from the leaf nodes of the tree and stops when it reaches the root, which means that the whole tree is encoded. Then two functions map the representation of the tree to the mean μ and the covariance σ of the multivariate normal Gaussian distribution that models the latent variables of the VAE. Finally, the so-called reparameterization trick [63] is performed to sample a latent vector of the VAE, see Figure 9 for a visual explanation of the encoding process.

More precisely, the tree encoding is defined as a function $\phi : \mathcal{L}(\hat{\mathcal{G}}) \rightarrow \mathbb{R}^{n_{\text{VAE}}}$ from a language $\mathcal{L}(\hat{\mathcal{G}})$ to a n_{VAE}-dimensional space. Consider a grammar rule of the form $\Psi \rightarrow s(L_1, ...L_k)$ that re-writes a non-terminal Ψ with a k-ary operation and produces $L_1, ..., L_k$ children non-terminals. For each grammar rule, we introduce a function $f^r : \mathbb{R}^{k \times n} \rightarrow \mathbb{R}^n$ (with n as a user-defined integer) that maps the representation of the k children to the representation of the parent. If the rule re-writes the non-terminal with a nullary operation, i.e. a constant, then f^r is a constant vector of size n. In practice f^r is defined as a fully connected neural network with a single layer:

$$f^r(t_1, ..., t_k) = \tanh(U^{r,j} t_j + b^r), \tag{12}$$

where $U^{r,j} \in \mathbb{R}^{n \times n}$ are the weights and $b \in \mathbb{R}^n$ the biases of the neural network. To obtain a representation of the whole tree, f^r is applied during parsing. Therefore, the tree encoder is defined as a recursive equation of the form:

$$\phi(s(\hat{t}_1, ...\hat{t}_k), \Psi) = f^r(\phi(\hat{t}_1, L_1), ..., \phi(\hat{t}_k, L_k)), \tag{13}$$

where $r \in R$ is the rule that derives $s(\hat{t}_1, ...\hat{t}_k)$ from Ψ. For the VAE, the encoding probability $q_\phi(\mathbf{z}|\hat{w})$, where $\hat{w}$ the derivation tree of an expression, is the Gaussian with mean $\mu : \mathbb{R}^n \to \mathbb{R}^{n_{\text{VAE}}}$ and the diagonal covariance $\sigma : \mathbb{R}^n \to \mathbb{R}^{n_{\text{VAE}}}$. Both μ and σ are defined as linear fully connected networks with trainable parameters. The mean and the covariance are then used to perform the reparameterization $\mathbf{z} = \mu + \epsilon \odot \sigma$, where $\epsilon \sim \mathcal{N}(0, \alpha \cdot \mathbf{I})$ and α positive real number that scales the covariance of ϵ. When the encoding finishes, the entire tree is represented by a vector $\mathbf{z} \in \mathbb{R}^{n_{\text{VAE}}}$.

Tree Decoder: The purpose of a tree decoder is to provide a process to generate trees conditioned on a low-dimensional vectorial representation. Even though the decoder and the generation process discussed in Appendix B both work in a top-down manner, initiating from the starting symbol, they present a notable difference. For each step l of the generation process, the generation algorithm recovers a non-terminal Ψ from the tail of the list of non-terminals, applies the rule $r_l = \Psi \to s(L_1, ...L_k)$, removes Ψ from the list and stores $L_1, ...L_k$, see Appendix B. In contrast to the generation process, the decoding process does not take a *list of grammar rules* as an input but a *vectorial encoding* $\mathbf{z}$. For this reason the rule r_l to be applied at the l-th step is not known a priori, but needs to be chosen during the tree generation.

The tree decoding is defined as the opposite process of the encoding, or more precisely, as a function $\psi : \mathbb{R}^{n_{\text{VAE}}} \to \mathcal{L}(\hat{\mathcal{G}})$ from a n_{VAE}-dimensional space to the language $\mathcal{L}(\hat{\mathcal{G}})$. Mirroring the encoder, the first step of the decoding process is to lift the dimensions of the vector $\mathbf{z} \in \mathbb{R}^{n_{\text{VAE}}}$ using a linear fully connected network $\rho : \mathbb{R}^{n_{\text{VAE}}} \to \mathbb{R}^n$ to obtain the vector $\mathbf{s} = \rho(\mathbf{z})$ that represents the entire tree. Then, at each step l of the decoding process, the algorithm takes a vector $\mathbf{s}$ and a non-terminal Ψ as inputs. The vectorial representation $\mathbf{s}$ is used to choose which rule r_l will be applied as follows. Let N_Ψ be the number of rules with Ψ on their left-hand side. A linear fully connected network $h_\Psi : \mathbb{R}^n \to \mathbb{R}^{N_\Psi}$ is employed to obtain a score vector $\lambda = h_\Psi(\mathbf{s})$ for each non-terminal Ψ. Then we choose a rule r_l, with $l \in [1, ..., N_\Psi]$, based on the probability:

$$p_\Psi(r_l|\mathbf{z}) = \frac{\exp(\lambda_l)}{\sum_{i=1}^{N_\Psi} \exp(\lambda_i)}. \tag{14}$$

For choosing the rules to be applied for non-terminals $L_1, ...L_k$, the representation of the parent needs to be mapped to the representation of the children. For this purpose, a neural network is defined with k layers $g_1^r, ..., g_k^r$, for each rule $r = \Psi \to s(L_1, ..., L_k)$, to map the vector representation $\mathbf{t}$ of each child, where $t_j = g_j^r(\mathbf{s})$ for $j \in [1, ..., k]$. At each step of the decoder, the tree representation $\mathbf{s}$ is updated by removing the representation at step l from the overall tree representation, i.e. $\mathbf{s}$ is substituted by $\mathbf{s} - \mathbf{t}_l$. We repeat this process until no non-terminal symbol is left. The decoding function is then written in a recursive manner as follows:

$$\psi(\mathbf{z}, \Psi) = s(\psi(\mathbf{t}_1, L_1), ..., \psi(\mathbf{t}_k, L_k)). \tag{15}$$

Using ψ for $\Psi = C$ where C is the starting symbol, we recursively generate a tree $\hat{w}$ given a vector $\mathbf{z}$. Even though the decoder learns probabilities of production rules conditioned to a library of valid material laws, which means that it is strongly biased to produce valid constitutive laws, it is not explicitly constrained to always satisfy the constitutive law constraints. This means that there exists a probability that the discovery process results to an expression that is not semantically valid.

Loss Function: The VAE introduces a probability density function $q_\phi(\mathbf{z}|\hat{w})$, parameterized by the encoder, and the conditional probability $p_\psi(\hat{w}|\mathbf{z})$ of decoding the input using the latent vector $\mathbf{z}$. The autoencoder is trained to minimize the loss:

$$\mathcal{L}(\phi, \psi) = \sum_{i=1}^{m} \mathbb{E}_{q_\phi(z_i|\hat{w}_i)}[-\log(p_\psi(\hat{w}_i|\mathbf{z}))] + \beta \mathcal{D}_{KL}(q_\phi||\mathcal{N}), \tag{16}$$

where $\mathcal{D}_{KL}(q_\phi||\mathcal{N})$ is the Kullback–Leibler divergence between the distribution parameterized by the encoder and the standard normal distribution $\mathcal{N}$ and β is a user-defined parameter that controls the effect of this term on the overall loss [64]. Eq. (16) trains the model to maximize the probability of choosing the correct rule for a step in the decoding process given that all the choices for the previous steps have been correct [42].

Inverse Problem Solution: The data-driven discovery stage of the process consists of searching the reduced space of the VAE for an encoding, which when decoded provides a model whose evaluation matches well the measured data. At this stage, we do not focus on the optimal solution of this problem, but rather opt for a heuristic method, which is a popular choice for the solution of inverse formulations linked to multi-step model evaluation processes [65]. We consider a global instead of a gradient based optimization method, because we expect the landscape of the latent space

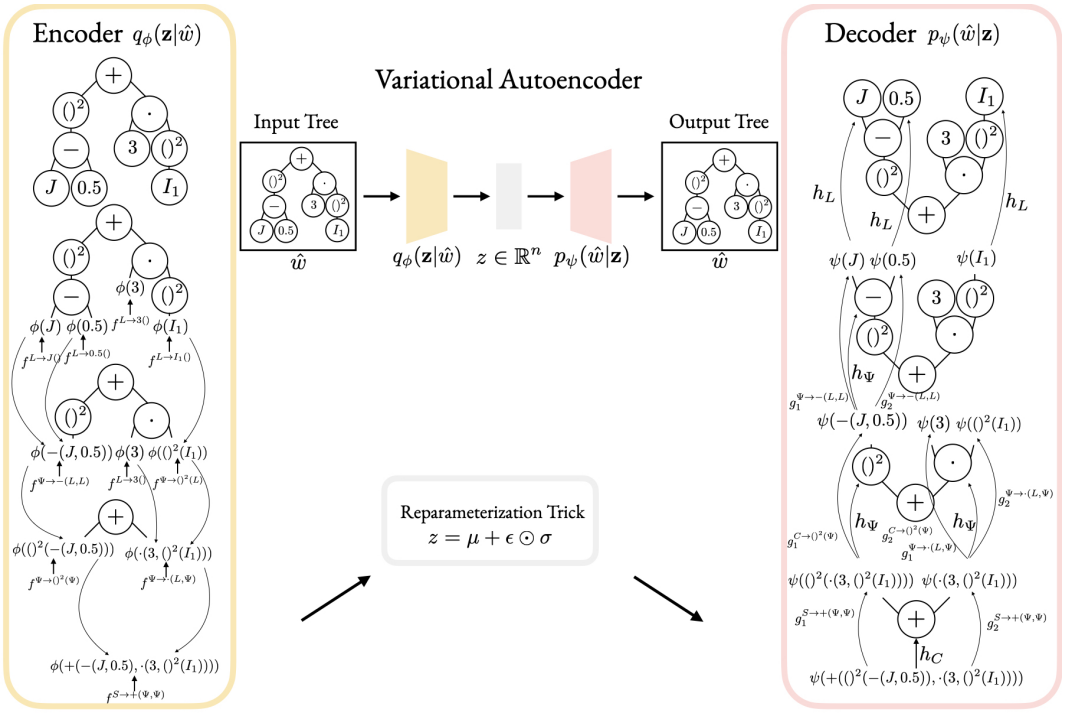


Figure 9: A schematic representation of the recursive tree VAE process for the case of $W(\mathbf{F}) = (J - 0.5)^2 + 3 \cdot \tilde{I}_1^2$. Left: The encoder maps the representation of the children to the representation of the parent using f^r, where r is the rule used to produce the children. The process starts from the leaf nodes until it reaches the root and encodes the whole tree. Middle: Perform the reparameterization trick to sample a latent vector $\mathbf{z}$. Right: The decoder generates a tree from a low-dimensional vector $\mathbf{z}$ by first using function h_Ψ to decide the next rule to be applied, beginning from the starting symbol, and then function g_j^r to predict the vectorial representations of the children.

to present multiple local minima. The data-driven discovery is performed on displacement measurements from only one type of loading (Biaxial Tension), which is not necessarily information to constrain the discovery to one material model, thus a global minimum. In this case, we opt for the adoption of a covariance matrix adaptation evolutionary strategy (CMA-ES) [59] for generating candidate solutions from a probability distribution whose parameters are iteratively updated. The CMA-ES method fits naturally for performing global optimization in the latent space of VAEs, as they both consider a multivariate normal distribution over the latent vectors $\mathbf{z}$. A difference lies in the fact that the CMA-ES uses a full covariance matrix during the adaptation process, while the VAE considers a diagonal covariance matrix. For measuring the goodness of fit, a root mean square error metric is adopted reflecting the discrepancy between the model evaluation and the available measurements. The CMA-ES reflects one of several possible heuristic tools one can opt for, with Genetic Algorithms [66] or Particle Swarm Optimization [67] representing further usual choices.

4.2 Constitutive Law Discovery Based on Data

In this section, we demonstrate how the proposed grammar-based approach can be used to perform efficient data-driven model discovery, thus covering the spectrum of tasks in Figure 1. The model discovery is *geometry and load agnostic*, which means that the same process without any change would work for any geometry, loading or boundary condition. We consider a simple RTG and generate a language for hyperelastic materials, which we then use to train a RTGVAE [42]. As a final step of the process we employ the CMA-ES [59] to discover the model that best fits synthetic data that are contaminated with different levels of noise. In real applications, these measurements may come from experiments or from higher fidelity (e.g., multiscale) finite element computations.

In the cases of EQL and SR, see Section 1, the data-driven discovery process is designed such that the learning algorithm chooses the combination that best fits the data out of a library of primitives, expressions or operations. The underlying assumption of these methods is that the constitutive laws can be described by a weighted sum of an priori chosen basis. The selection of such a basis biases the discovery process to recover combinations of already known relations and it is conditioned by the expertise of the person designing the library. However, in real applications the form of the constitutive law is not always known a priori such that an informed choice of the basis can be made. For such

cases, a weighted sum of the library elements may not accurately describe the constitutive law, because the library construction may be missing important information. With our grammar-based approach, on one hand we can fully automatically generate a much wider library of material models than possible by manual construction, as already demonstrated in Section 3.5; on the other hand, we can discover an interpretable material model which well interprets the data even though the true model underlying the data is not present in the library. While the capability of finding a good approximation for a model not present in the library was already demonstrated with SR [2], it still had to be a combination of the library terms. This is no longer the case in the present setting, as we demonstrate in this section.

Data Generation. For generating the data that we are going to use to perform the discovery, we consider the benchmark proposed in [2, 68] that emulates digital image correlation measurements using computational data generated by the finite element method. The domain is a plate with a central hole which undergoes a symmetric bi-axial loading controlled by a displacement parameter δ. Due to symmetry, only a quarter of the plate is studied with symmetry boundary conditions on the left and bottom boundaries [2], see Figure 6. Plane-strain conditions are assumed. In the data generation, we consider four different (baseline) hyperelastic constitutive models:

- A Neo-Hookean (NH) model containing a quadratic volumetric term

$$W = 0.5 \cdot (\tilde{I}_1 - 3) + 1.5 \cdot (J-1)^2;$$

- An Isihara (IS) model [69] containing a quadratic deviatoric term

$$W = 0.5 \cdot (\tilde{I}_1 - 3) + (\tilde{I}_2 - 3) + (\tilde{I}_1 - 3)^2 + 1.5 \cdot (J-1)^2;$$

- A Haines-Wilson (HW) model [70] containing a cubic deviatoric term

$$W = 0.5 \cdot (\tilde{I}_1 - 3) + (\tilde{I}_2 - 3) + 0.7 \cdot (\tilde{I}_1 - 3) \cdot (\tilde{I}_2 - 3) + 0.2 \cdot (\tilde{I}_1 - 3)^3 + 1.5 \cdot (J-1)^2;$$

- A Gent-Thomas (GT) model [71] containing a logarithmic deviatoric term

$$W = 0.5 \cdot (\tilde{I}_1 - 3) + \log(\tilde{I}_2/3) + 1.5 \cdot (J-1)^2.$$

We set the applied displacement parameter to $\delta = 0.3$ and, for each constitutive model, we obtain the displacement field $\mathbf{u}$ which we then use to compute the deformation gradient $\mathbf{F}$ and the volumetric and isochoric invariants of the right Cauchy-Green deformation tensor $J, \tilde{I}_1, \tilde{I}_2$. We add artificial noise to the displacement data, as follows:

$$\tilde{u}_i^a = u_i^a + e_i^a, \quad e_i^a \sim \mathcal{N}(0, \sigma^*) \quad \forall i = 1, 2, \forall a = 1, \dots, n_{dofs}, \tag{17}$$

with n_{dofs} as the number of degrees of freedom in the finite element discretization. We adopt two levels of noise, namely $\sigma^* = 10^{-4}$, and $\sigma^* = 10^{-3}$, as in [2]. We perform no denoising on the resulting displacement data $\tilde{\mathbf{u}}$.

Choosing the Grammar. For the reasons explained in Section 2.3, we adopt a RTG for the data-driven discovery of the material model. With respect to the grammars we used so far (e.g. the CFG $\mathcal{G}_{\text{HMp}}$ in eq. (11)), we introduce some simplifications; namely, we consider the terminals $J-1$, $\tilde{I}_1 - 3$, and $\tilde{I}_2 - 3$ to promote expressions where the normalization holds a priori. However the normalization conditions are not strictly enforced, as the final expression could contain operations that violate them, e.g. additions between scalars. Moreover, we rewrite C as $\Psi^1 + \Psi^2 + \Psi^3$ to derive final expressions that contain a smaller number of operations such that the grammar derives parsimonious expressions. The designed RTG, $\hat{\mathcal{G}}_{\text{HMs}}$, is defined as follows:

$$
\begin{aligned}
&S = \{\, C \,\},\\
&\tilde{\Sigma} = \{\, +:2,\ /:2,\ \cdot:2, (\cdot)^2:1,\ (\cdot)^3:1,\ \log:1,\ (\tilde{I}_1-3):0,\ (\tilde{I}_2-3):0,\\
&\qquad (J-1):0,\ 0.2:0,\ 0.5:0,\ 0.7:0,\ 1.5:0,\ 2:0,\ 3:0\},\\
&R = \{C \rightarrow\ \Psi^1+\Psi^2+\Psi^3,\\
&\qquad \Psi^1 \rightarrow L|U|Y|W|Q,\\
&\qquad \Psi^2 \rightarrow L|U|Y|W|Q,\\
&\qquad \Psi^3 \rightarrow L|U|Y|W|Q,\\
&\qquad Y \rightarrow\ +(L,L) \mid /(L,L) \mid\ \cdot\,(L,L)\ ,\\
&\qquad U \rightarrow \log(L),\\
&\qquad W \rightarrow\ ()^2(Y) \mid ()^3(Y),\\
&\qquad E \rightarrow\ \cdot(Y,Y),\\
&\qquad L \rightarrow V\,|O,\\
&\qquad V \rightarrow (\tilde{I}_1-3)() \mid (\tilde{I}_2-3)() \mid (J-1)(),\\
&\qquad O \rightarrow\ 0.2() \mid 0.5() \mid 0.7() \mid 1.5()\},\\
&\Phi = \{\, C, \Psi^1, \Psi^2, \Psi^3, Y, U, E, W, L, V, O \,\}.
\end{aligned}
\tag{18}
$$

In eq. (18), as done previously, we use the symbol "|" to separate grammar rules with the same left-hand side. The grammar $\hat{\mathcal{G}}_{\text{HMs}}$ contains additions and multiplications between terminals and thus it can potentially derive the NH model. However, the $\log(\tilde{I}_2/3)$ operation of the GT model or additional $+$ operations for the IS and HW models cannot be used when deriving the library of constitutive laws, hence these constitutive laws cannot be recovered exactly. Thus, for the data obtained using these laws our goal is to discover parsimonious expressions that deliver a physical behavior similar to the baseline using the available grammar $\hat{\mathcal{G}}_{\text{HMs}}$.

Training Setup. We derive expressions from the language $\mathcal{L}(\hat{\mathcal{G}}_{\text{HMs}})$ and construct a library of $N_{\text{train}} = 600,000$ valid constitutive models for different hyperelastic materials. We then train the recursive tree VAE model on this library. For the VAE, we set $\beta = 0.0001$ in eq. (16) and $\alpha = 0.0001$ for the reparameterization trick, see Section 4.1. We choose the latent dimension of the VAE as $n = 8$, the learning rate as 0.001 and the width of the recursive encoder and decoder as 128. We check the performance of the model against $N_{\text{test}} = 100$ test expressions using a root mean tree edit distance metric [72].

Data-driven Discovery. Once we train the model on the library, we perform gradient-free optimization on the latent space of the RTGVAE to discover different hyperelastic material models using the CMA-ES method. For CMA-ES we consider the number of new solutions per iteration as $1,000$ trees, a zero mean, a 0.1 variance and 10 maximum iterations per adaptation. Since dense (full-field) displacement measurements are available and the expression of the baseline constitutive law is known, we can assess the goodness of fit using a root mean square error metric:

$$
\text{RMSE}_W = \sqrt{\frac{1}{N_n}\sum_{i=1}^{N_n}(W_i - \tilde{W}_i)^2},
\tag{19}
$$

where W_i and $\tilde{W}_i$ are respectively the baseline and the predicted strain energies for node i, and $N_n = 2,752$ is the number of nodes in the non-uniform finite element mesh used for data generation. The computation of the root mean square error is performed using the SymEngine [73] SymPy interface for fast computations.

For the assessement of the prediction accuracy, we employ the relative $\mathcal{L}^2$ error metric referred to the entire domain:

$$
\text{Relative } \mathcal{L}^2 \text{ Error} = \frac{\|W - \tilde{W}\|_2}{\|W\|_2}
$$

and analogous for other quantities, such as the first component of the first Piola-Kirchhoff stress tensor.

In Table 1, we report on the wall-clock time to run an iteration of the CMA-ES method with respect to the new solutions generated at each adaptation step on an Alienware m16 Laptop with an Intel i9-13900HX CPU and 32GB RAM.

New solutions per iteration	100	500	1,000	5,000
Wall-clock time in sec	1	5	11	60

Table 1: The wall-clock time in seconds of the discovery process for different numbers of new solution per iteration that are used for the covariance adaptation.

Assessing the Discovery Process for the NH Model. First, we test the performance of the symbolic regression algorithm in discovering the NH model from displacement data with $\sigma^* = 0, 10^{-4}, 10^{-3}$. The best discovered expressions as well as the relative $\mathcal{L}^2$ errors between the baseline and the predicted strain energy density are presented in Table 2 for different noise levels. Being contained in the language generated by the starting grammar, the NH model is discovered *exactly* for the noise-free case. Interestingly, the process is able to discover the exact model also for the noisy case with $\sigma^* = 10^{-4}$, although no denoising is applied to the data. For the $\sigma^* = 10^{-3}$ case, a very accurate approximation is discovered, with a relative $\mathcal{L}^2$ error of 1.2%.

Model	Discovered Expression	Relative $\mathcal{L}^2$ Error
NH, $\sigma^* = 0$	$\tilde{W} = 0.5 \cdot (I_1 - 3) + 1.5 \cdot (J-1)^2$	0
NH, $\sigma^* = 10^{-4}$	$\tilde{W} = 0.5 \cdot (I_1 - 3) + 1.5 \cdot (J-1)^2$	0
NH, $\sigma^* = 10^{-3}$	$\tilde{W} = 0.49 \cdot (I_1 - 3) + 1.5 \cdot (J-1)^2$	0.012

Table 2: Best discovered expressions and relative $\mathcal{L}^2$ errors between the best predicted and the baseline strain energy density for the NH model, obtained from data with different noise levels.

Assessing the Discovery Process for the IS, HW and GT Models. We then run the discovery algorithm for each of the IS, HW and GT models separately. Since none of these models is contained in the language generated by the grammar in use, we do not expect an exact discovery. The best discovered models for the noise-free, $\sigma^* = 10^{-4}$, and $\sigma^* = 10^{-3}$ noise cases as well as the relative $\mathcal{L}^2$ errors between the baseline and the predicted strain energy density are presented in Table 3. We observe that for the IS and the HW models the relative $\mathcal{L}^2$ error is less than 1.5% for all noise levels, while for the GT model the error increases to 2.6% for all cases.

We note that the models discovered from the IS data are very close to the baseline ones. While in the grammar (18) we defined the constitutive law as $\Psi^1 + \Psi^2 + \Psi^3$, which means that constructed expressions in the library contain two binary $+$ operations, the discovery process returns an expression containing three binary operations; this indicates that the decoding process can predict expressions that are not limited to the strict structure of those contained within the library. Thus, interestingly, the discovered models for the noiseless and the lowest-noise case have the same structure of the baseline model, but different material constants. We attribute the discrepancy in the constants to inaccuracies of the symbolic regression algorithm and the inability of the heuristic CMA-ES method to determine the global minimum of the optimization problem. Also interestingly, the predicted expression for the HW model is less complicated than the baseline and yet it achieves an excellent accuracy, with an error less than 1% independently of the noise level. Finally, for the GT model the discovered expression is a linearized version of the baseline, which is reasonable for the relatively limited amount of deformation included in the data (a similar result is reported in [2]). This further reflects the dependence of the inference process on the richness of the available dataset, including the amount of non-linearity that is observed.

Model	Discovered Expression	Relative $\mathcal{L}^2$ Error
IS, $\sigma^* = 0$	$\tilde{W} = \tilde{I}_1 - 3 + 0.5 \cdot (\tilde{I}_2 - 3) + (\tilde{I}_1 - 3)^2 + 1.5 \cdot (J-1)^2$	0.011
IS, $\sigma^* = 10^{-4}$	$\tilde{W} = \tilde{I}_1 - 3 + 0.5 \cdot (\tilde{I}_2 - 3) + (\tilde{I}_1 - 3)^2 + 1.5 \cdot (J-1)^2$	0.011
IS, $\sigma^* = 10^{-3}$	$W = (\tilde{I}_1 - 3) \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (\tilde{I}_1 - 3) + 1.5 \cdot (J-1)^2$	0.014
HW, $\sigma^* = 0$	$\tilde{W} = 1.5 \cdot (\tilde{I}_2 - 3) + (\tilde{I}_1 - 3)^2 + 1.5 \cdot (-1 + J)^2$	0.01
HW, $\sigma^* = 10^{-4}$	$\tilde{W} = (\tilde{I}_1 - 3) \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (-1 + J)^2$	0.007
HW, $\sigma^* = 10^{-3}$	$\tilde{W} = (\tilde{I}_1 - 3) \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (-1 + J)^2$	0.007
GT, $\sigma^* = 0$	$\tilde{W} = 0.5 \cdot (\tilde{I}_1 - 3) + 0.3 \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (J-1)^2$	0.026
GT, $\sigma^* = 10^{-4}$	$W = 0.5 \cdot (\tilde{I}_1 - 3) + 0.3 \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (J-1)^2$	0.026
GT, $\sigma^* = 10^{-3}$	$\tilde{W} = 0.5 \cdot (\tilde{I}_1 - 3) + 0.2 \cdot (\tilde{I}_2 - 3) + 1.5 \cdot (J-1)^2$	0.026

Table 3: Best discovered expressions and relative $\mathcal{L}^2$ errors between the best predicted and the baseline strain energy density for the IS, HW and GT models, obtained from data with different noise levels.

Generalization to Different Loading Conditions. In practice, we are interested in expressions that describe the behavior of a material under general loading conditions. For this reason, we now assess the performance of the discovered models in an extrapolation setup (i.e. for problems different from the one in Figure 6). For each discovered model, we perform the six simple loading tests in equation 7 in the range $\gamma \in [0, 1]$. We compare the baseline and the discovered constitutive laws in terms of strain energy density and first component of the first Piola-Kirchhoff stress tensor for all six loading conditions and noise levels. We present the results for the NH, IS, HW and GT models in Figures 10, 11, 12, and 13, respectively.

As expected, the NH model predictions are identical to the baseline for the noise-free and $\sigma^* = 10^{-4}$ cases, while they are very close to the baseline for $\sigma^* = 10^{-3}$. For the IS model, the predictions for the strain energy density in the noise-free case are very close to the baseline, while for the noisy cases we observe discrepancies especially in UC. The predictions for the first component of the first Piola-Kirchhoff stress tensor are very accurate for the noise-free case, but present discrepancies for the noisy cases especially for SS and UC loading. For the HW model, the predictions for the strain energy density are quite accurate for all loading conditions and noise levels, except for UC and BC, where we observe discrepancies for higher deformations. For the first component of the first Piola-Kirchhoff, we observe more pronounced discrepancies especially for the UC, SS and PS loading conditions and in the presence of noise. For the GT model, we observe a very good agreement between the baseline and the predicted model for all loading conditions for both strain energy density and stress in the noise-free and $\sigma^* = 10^{-4}$ cases. However, for the noise level $\sigma^* = 10^{-3}$ we observe high discrepancies for most loading conditions, which is reasonable considering the significant level of noise contamination and the usage of the raw noise-affected data with no denoising. For all the models and out of all loading cases, slightly larger discrepancies are obtained for the loading conditions involving shear. This can be explained by the fact that the extent of shear in the training data is quite limited. Similar observations were made in previous works with model discovery approaches based on SR [2].

5 Conclusions

In this paper, we proposed formal grammars (specifically, Context-Free Grammars and Regular Tree Grammars) as conceptual and practical tools for the automated discovery of material models, featuring low complexity, ability to systematically embed constraints stemming from domain knowledge, and generalization capability. Leveraging formal grammars specifically designed for constitutive laws (focusing on hyperelasticity in this initial paper), we could achieve two important goals. On the one hand, we could automatically generate extensive libraries of hyperelastic material models including a desired, very large number of terms which satisfy the relevant physics constraints, combine high expressivity with parsimony and can be deployed for model discovery based on sparse regression. On the other hand, we proposed and explored a grammar-based symbolic regression pipeline for constitutive law discovery composed by a library construction step, a pre-training step, and a data-driven discovery step, and leveraging a combination of the Recursive Tree Grammar Variational Autoencoder method [42] and the Covariant Matrix Adaptation Evolutionary Strategy [59] for gradient-free optimization. We demonstrated that the approach is able to discover accurate and parsimonious hyperelastic models on four different benchmark cases.

We here put forth an important first step toward establishing the language of material models, along with a number of possible useful downstream tasks and applications. However, the proposed method in its current form presents certain limitations. First, the efficiency and computational wall clock time required for the library generation is limited by the rate of rejections of the expressions, because the normalization and volumetric growth constraints are not imposed a priori but checked after the final expression is generated. Moreover, the decoder of the VAE does not allow to check the semantic validity of the generated expression during the process of decoding, which implies that expressions are yielded that may not satisfy the constitutive law constraints. Furthermore, the CMA-ES method is an evolutionary strategy, which implies that different runs of CMA-ES can lead to different results and expressions offering different levels of approximation to the baseline. Finally, in this paper we considered grammars that only describe the behavior of isotropic hyperelastic materials, but the proposed methodology is extendable to more general and complex classes of material behavior. These limitations will be addressed in forthcoming research.

Code Availability

The code for automated model discovery with the proposed procedure will be made available online at the time of publication.

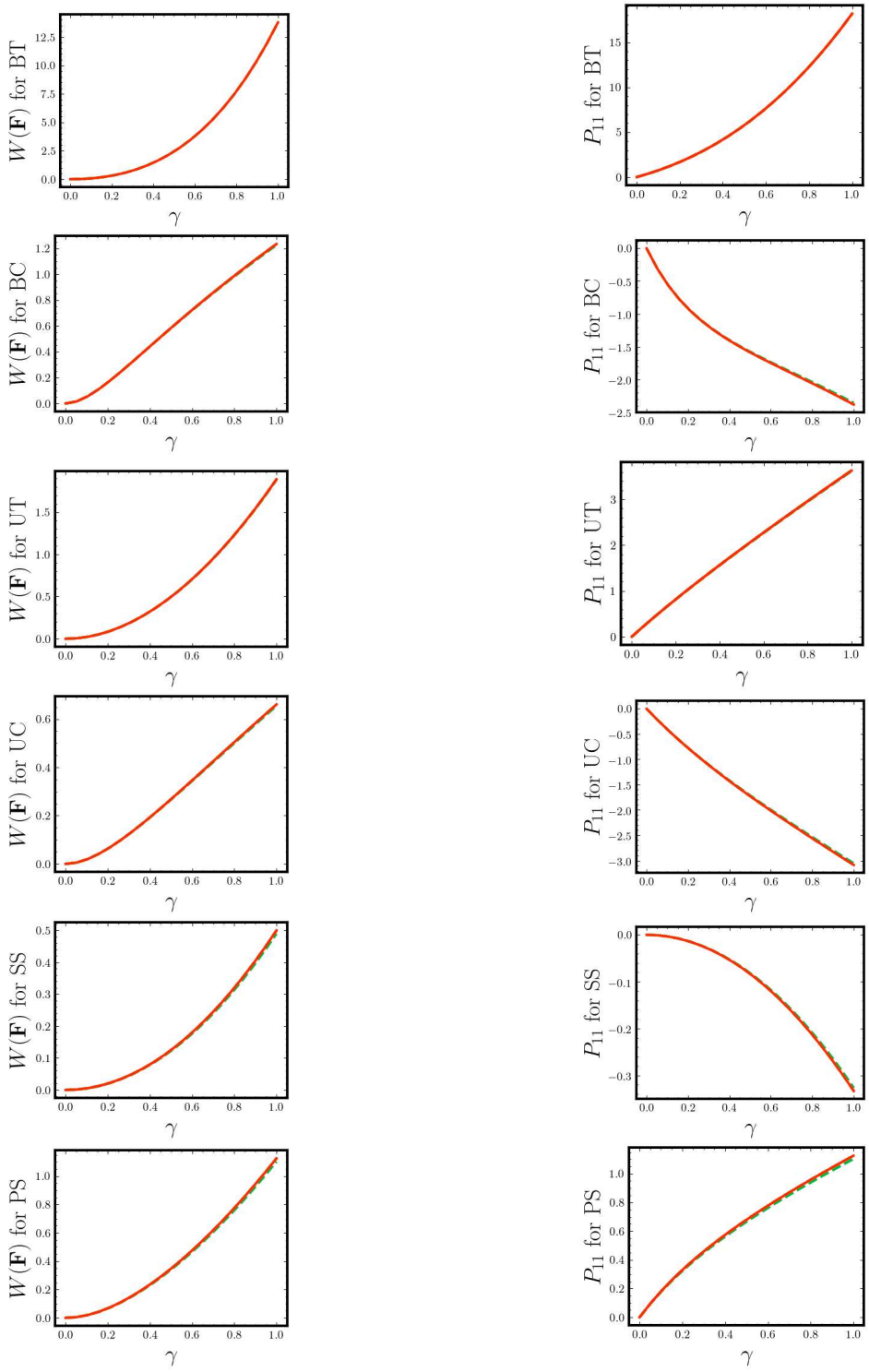


Figure 10: Neo-Hookean model: Left: comparison between the baseline and the predicted strain energy density, Right: comparison between the first component of the Piola-Kirchhoff stress tensor for the loading conditions not in the training set evaluated for different displacement magnitudes parameterized by γ. For all cases, the solid red line represents the baseline, the dashed blued line the prediction for the noise-free data, the orange dashed line the prediction for $\sigma^* = 10^{-4}$ and the green dashed line the prediction for $\sigma^* = 10^{-3}$.

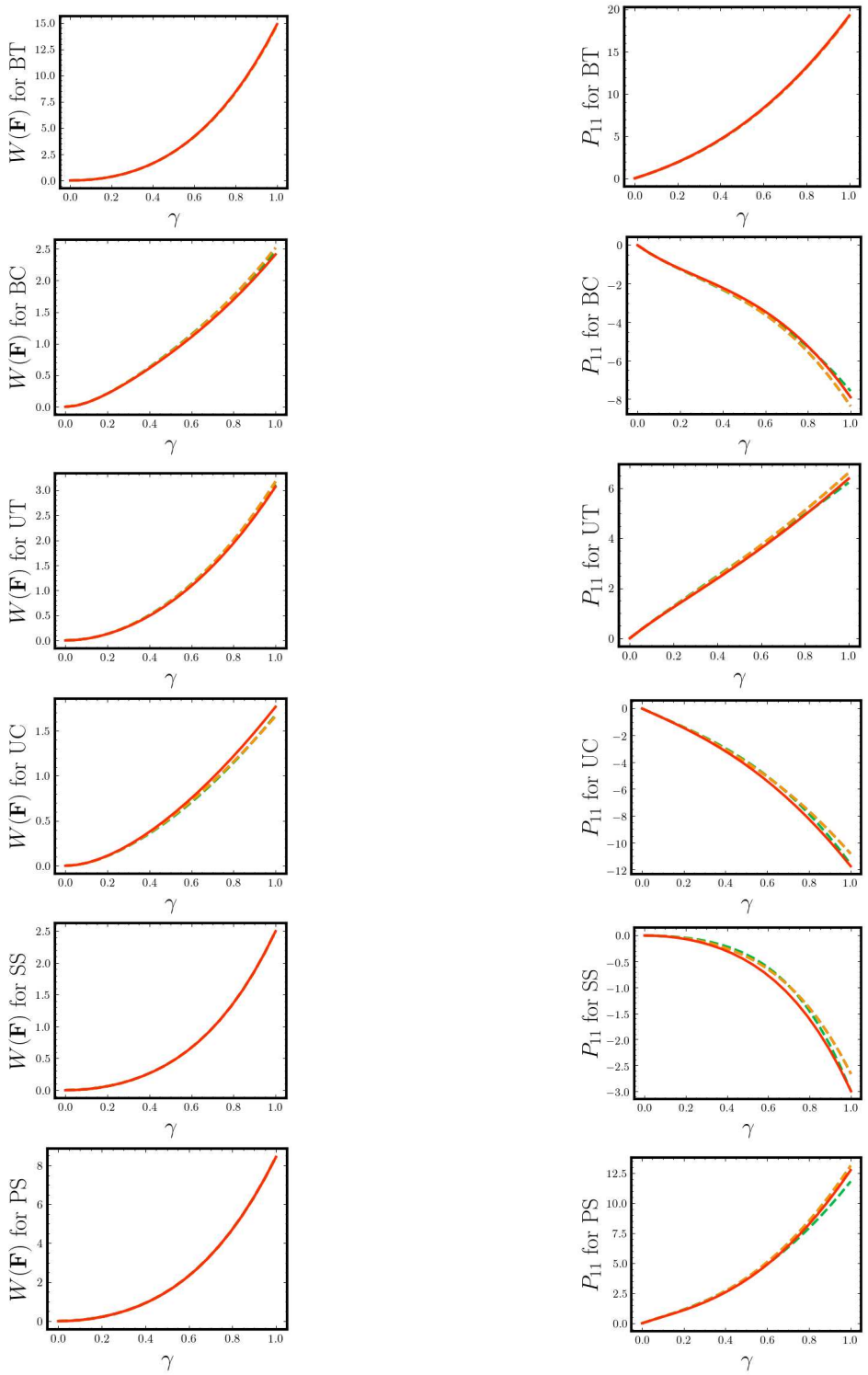


Figure 11: Isihara model: Left: comparison between the baseline and the predicted strain energy density, Right: comparison between the first component of the Piola-Kirchhoff stress tensor for the loading conditions not in the training set evaluated for different displacement magnitudes parameterized by γ. For all cases, the solid red line represents the baseline, the dashed blued line the prediction for the noise-free data, the orange dashed line the prediction for $\sigma^* = 10^{-4}$ and the green dashed line the prediction for $\sigma^* = 10^{-3}$.

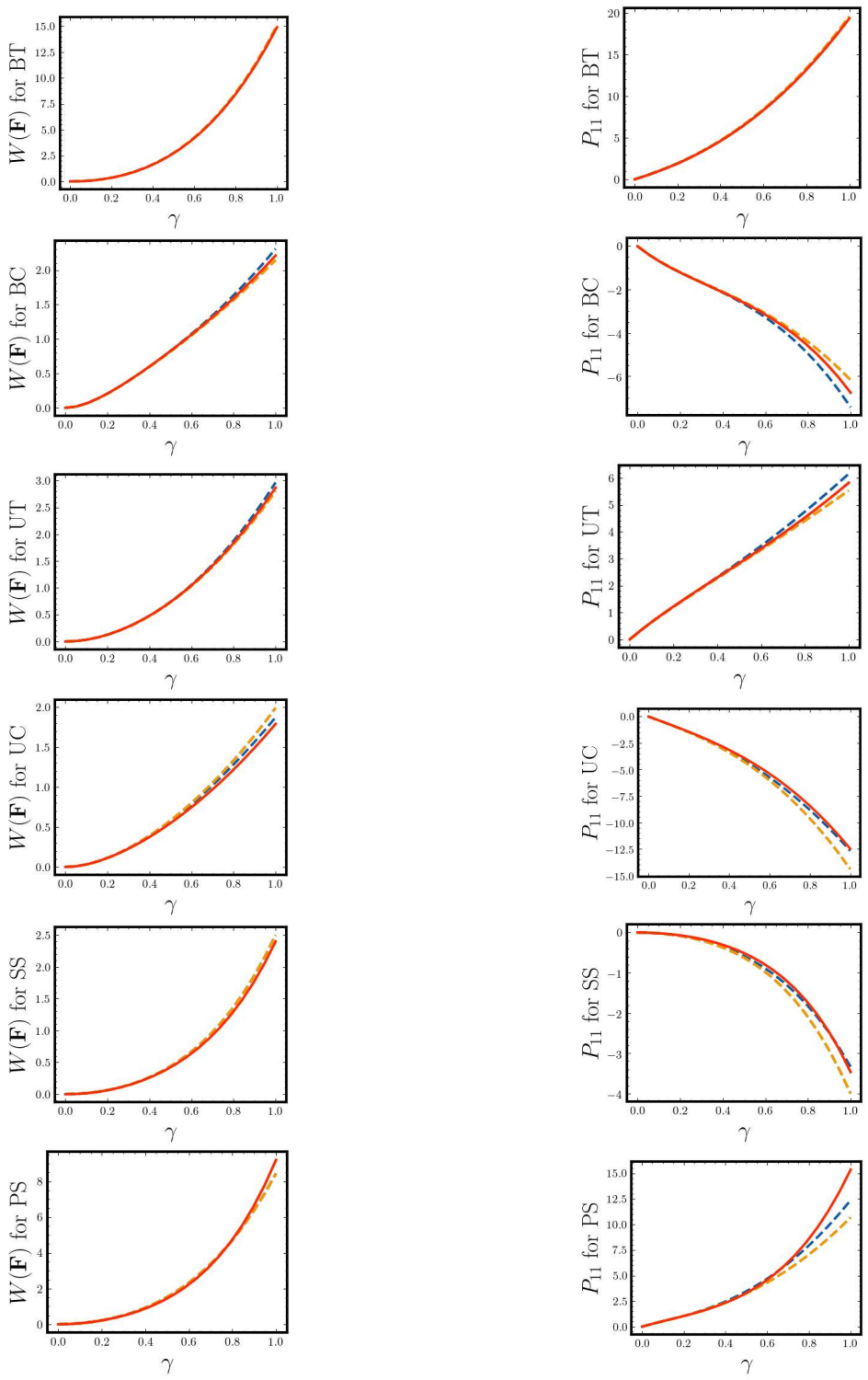


Figure 12: Haines-Wilson model: Left: comparison between the baseline and the predicted strain energy density, Right: comparison between the first component of the Piola-Kirchhoff stress tensor for the loading conditions not in the training set evaluated for different displacement magnitudes parameterized by γ. For all cases, the solid red line represents the baseline, the dashed blued line the prediction for the noise-free data, the orange dashed line the prediction for $\sigma^* = 10^{-4}$ and the green dashed line the prediction for $\sigma^* = 10^{-3}$.

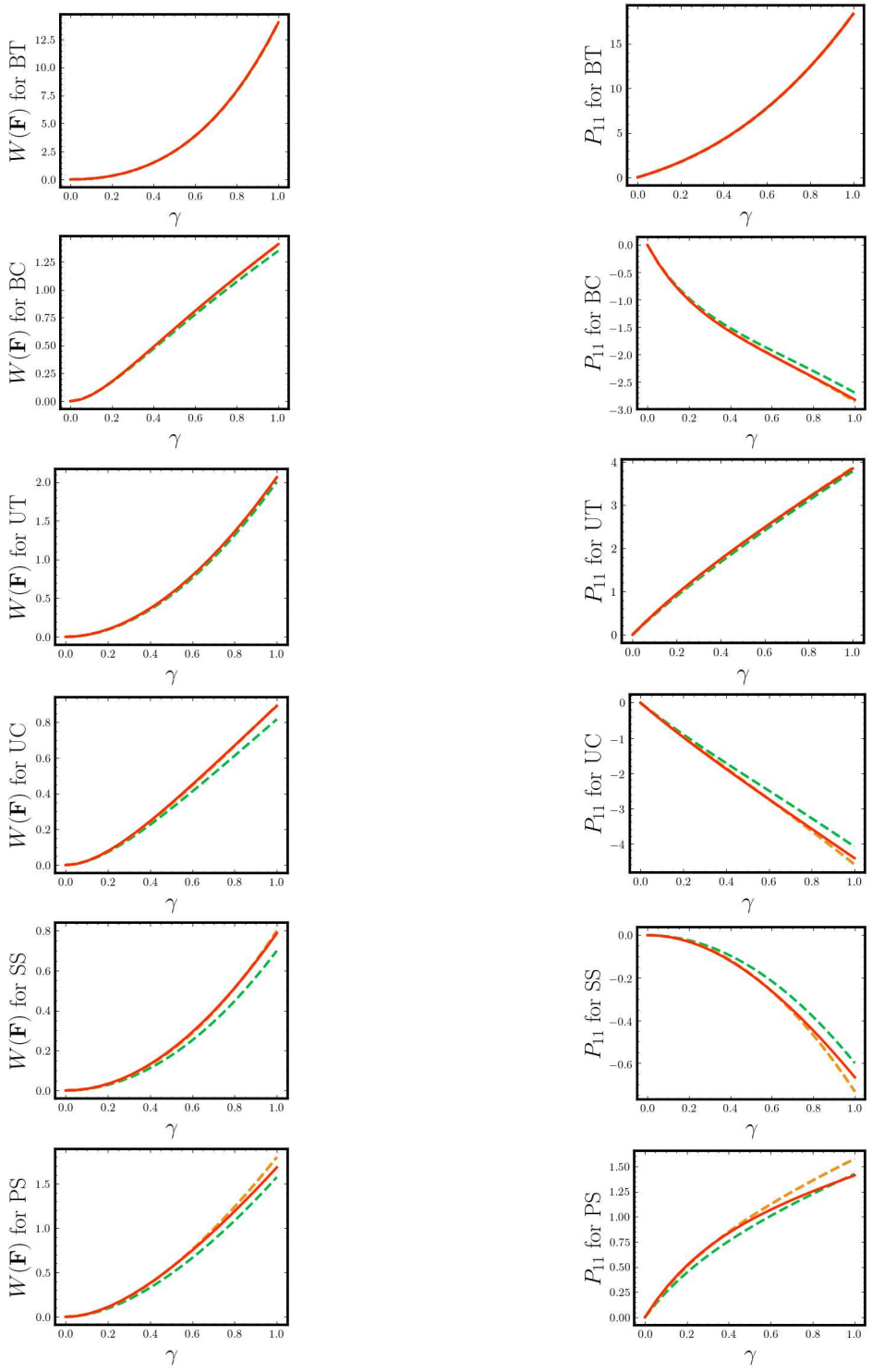


Figure 13: Gent-Thomas model: Left: comparison between the baseline and the predicted strain energy density, Right: comparison between the first component of the Piola-Kirchhoff stress tensor for loading conditions not in the training set evaluated for different displacement magnitudes parameterized by γ. For all cases, the solid red line represents the baseline, the dashed blued line the prediction for the noise-free data, the orange dashed line the prediction for $\sigma^* = 10^{-4}$ and the green dashed line the prediction for $\sigma^* = 10^{-3}$.

Acknowledgments

G.K. would like to acknowledge support from Asuera Stiftung via the ETH Zurich Foundation. G.K. would like to thank Anej Svete and Konstantinos Kallas for the fruitful discussions regarding formal languages. L.D.L. acknowledges funding by the Swiss National Science Foundation through grant N. 200021-204316 "Unsupervised data-driven discovery of material laws". S.M. acknowledges support in part by the DOE SEA-CROGS project (DE-SC0023191).

References

[1] Rolf Mahnken. Identification of material parameters for constitutive equations. *Encyclopedia of computational mechanics*, 2004.

[2] Moritz Flaschel, Siddhant Kumar, and Laura De Lorenzis. Unsupervised discovery of interpretable hyperelastic constitutive laws. *Computer Methods in Applied Mechanics and Engineering*, 381:113852, 2021.

[3] Hugo Luiz Oliveira, François Louf, and Fabrice Gatuingt. Numerical study based on the constitutive relation error for identifying semi-rigid joint parameters between planar structural elements. *Engineering Structures*, 236:112015, 2021.

[4] Yoh-ichi Mototake, Hitoshi Izuno, Kenji Nagata, Masahiko Demura, and Masato Okada. A universal bayesian inference framework for complicated creep constitutive equations. *Scientific Reports*, 10(1):10437, 2020.

[5] F Pierron and M Grédiac. Towards material testing 2.0. a review of test design for identification of constitutive parameters from full-field measurements. *Strain*, 57(1):e12370, 2021.

[6] Liang Li, Qian Shao, Yichen Yang, Zengtao Kuang, Wei Yan, Jie Yang, Ahmed Makradi, and Heng Hu. A database construction method for data-driven computational mechanics of composites. *International Journal of Mechanical Sciences*, 249:108232, 2023.

[7] Frederic E Bock, Roland C Aydin, Christian J Cyron, Norbert Huber, Surya R Kalidindi, and Benjamin Klusemann. A review of the application of machine learning and data mining approaches in continuum materials mechanics. *Frontiers in Materials*, 6:110, 2019.

[8] Trenton Kirchdoerfer and Michael Ortiz. Data-driven computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 304:81–101, 2016.

[9] Georg Martius and Christoph H Lampert. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.

[10] Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pages 4442–4450. PMLR, 2018.

[11] Allan Costa, Rumen Dangovski, Owen Dugan, Samuel Kim, Pawan Goyal, Marin Soljačić, and Joseph Jacobson. Fast neural models for symbolic regression at scale. *arXiv preprint arXiv:2007.10784*, 2020.

[12] Kevin Linka and Ellen Kuhl. A new family of constitutive artificial neural networks towards automated model discovery. *Computer Methods in Applied Mechanics and Engineering*, 403:115731, 2023.

[13] Vahidullah Taç, Kevin Linka, Francisco Sahli-Costabal, Ellen Kuhl, and Adrian Buganza Tepole. Benchmarking physics-informed frameworks for data-driven hyperelasticity. *Computational Mechanics*, pages 1–17, 2023.

[14] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[15] Ryan J Tibshirani. The lasso problem and uniqueness. 2013.

[16] Hui Zou, Trevor Hastie, and Robert Tibshirani. On the "degrees of freedom" of the lasso. *The Annals of Statistics*, 35(5):2173–2192, 2007.

[17] Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K Petersen. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35:33985–33998, 2022.

[18] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[19] Akshay Joshi, Prakash Thakolkaran, Yiwen Zheng, Maxime Escande, Moritz Flaschel, Laura De Lorenzis, and Siddhant Kumar. Bayesian-euclid: Discovering hyperelastic material laws with uncertainties. *Computer Methods in Applied Mechanics and Engineering*, 398:115225, 2022.

[20] Moritz Flaschel, Huitian Yu, Nina Reiter, Jan Hinrichsen, Silvia Budday, Paul Steinmann, Siddhant Kumar, and Laura De Lorenzis. Automated discovery of interpretable hyperelastic material models for human brain tissue with euclid. *Journal of the Mechanics and Physics of Solids*, 180:105404, 2023.

[21] Jagannadh Boddapati, Moritz Flaschel, Siddhant Kumar, Laura De Lorenzis, and Chiara Daraio. Single-test evaluation of directional elastic properties of anisotropic structured materials. *Journal of the Mechanics and Physics of Solids*, 181:105471, 2023.

[22] Sarah R St Pierre, Kevin Linka, and Ellen Kuhl. Principal-stretch-based constitutive neural networks autonomously discover a subclass of ogden models for human brain tissue. *Brain Multiphysics*, 4:100066, 2023.

[23] Skyler R St Pierre, Divya Rajasekharan, Ethan C Darwin, Kevin Linka, Marc E Levenston, and Ellen Kuhl. Discovering the mechanics of artificial and real meat. *Computer Methods in Applied Mechanics and Engineering*, 415:116236, 2023.

[24] Enzo Marino, Moritz Flaschel, Siddhant Kumar, and Laura De Lorenzis. Automated identification of linear viscoelastic constitutive laws with euclid. *Mechanics of Materials*, 181:104643, 2023.

[25] Moritz Flaschel, Siddhant Kumar, and Laura De Lorenzis. Discovering plasticity models without stress data. *npj Computational Materials*, 8(1):91, 2022.

[26] Bahador Bahmani, Hyoung Suk Suh, and WaiChing Sun. Discovering interpretable elastoplasticity models via the neural polynomial method enabled symbolic regressions. *arXiv preprint arXiv:2307.13149*, 2023.

[27] Moritz Flaschel, Siddhant Kumar, and Laura De Lorenzis. Automated discovery of generalized standard material models with euclid. *Computer Methods in Applied Mechanics and Engineering*, 405:115867, 2023.

[28] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4:87–112, 1994.

[29] T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

[30] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[31] Marco Virgolin and Solon P Pissis. Symbolic regression is np-hard. *arXiv preprint arXiv:2207.01018*, 2022.

[32] Surajit Pal, G Wije Wathugala, and Sukhamay Kundu. Calibration of a constitutive model using genetic algorithms. *Computers and Geotechnics*, 19(4):325–348, 1996.

[33] Donovan Birky, Karl Garbrecht, John Emery, Coleman Alleman, Geoffrey Bomarito, and Jacob Hochhalter. Generalizing the gurson model using symbolic regression and transfer learning to relax inherent assumptions. *Modelling and Simulation in Materials Science and Engineering*, 2023.

[34] Bahador Bahmani and WaiChing Sun. Physics-constrained symbolic model discovery for polyconvex incompressible hyperelastic materials. *arXiv preprint arXiv:2310.04286*, 2023.

[35] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[36] Brenden K Petersen, Claudio P Santiago, and Mikel Landajuela. Incorporating domain knowledge into neural-guided search. *arXiv preprint arXiv:2107.09182*, 2021.

[37] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and Francois Charton. End-to-end symbolic regression with transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[38] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. PMLR, 2021.

[39] Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. Symformer: End-to-end symbolic regression using transformer-based architecture. *arXiv preprint arXiv:2205.15764*, 2022.

[40] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International conference on machine learning*, pages 1945–1954. PMLR, 2017.

[41] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *International Conference on Learning Representations*, 2018.

[42] Benjamin Paaßen, Irena Koprinska, and Kalina Yacef. Recursive tree grammar autoencoders. *Machine Learning*, 111(9):3393–3423, 2022.

[43] Hendrik Jan Hoogeboom. Undecidable problems for context-free grammars. *Preprint https://liacs. leidenuniv. nl/˜ hoogeboomhj/second/codingcomputations. pdf*, 2015.

[44] Peter Linz and Susan H Rodger. *An introduction to formal languages and automata*. Jones & Bartlett Learning, 2022.

[45] Javier Bonet and Richard D Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press, 1997.

[46] Gerhard A Holzapfel. Nonlinear solid mechanics: a continuum approach for engineering science, 2002.

[47] John M Ball. Convexity conditions and existence theorems in nonlinear elasticity. *Archive for rational mechanics and Analysis*, 63:337–403, 1976.

[48] Stefan Hartmann and Patrizio Neff. Polyconvexity of generalized polynomial-type hyperelastic strain energy functions for near-incompressibility. *International journal of solids and structures*, 40(11):2767–2791, 2003.

[49] Akshay Joshi, Prakash Thakolkaran, Yiwen Zheng, Maxime Escande, Moritz Flaschel, Laura De Lorenzis, and Siddhant Kumar. Bayesian-euclid: Discovering hyperelastic material laws with uncertainties. *Computer Methods in Applied Mechanics and Engineering*, 398:115225, 2022.

[50] Jure Brence, Ljupčo Todorovski, and Sašo Džeroski. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224:107077, 2021.

[51] Frederick Jelinek, John D Lafferty, and Robert L Mercer. *Basic methods of probabilistic context free grammars*. Springer, 1992.

[52] Zhiyi Chi. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160, 1999.

[53] Stuart Geman and Mark Johnson. Probabilistic grammars and their applications. *International Encyclopedia of the Social & Behavioral Sciences*, 2002:12075–12082, 2002.

[54] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[55] Joze Korelc and Peter Wriggers. *Automation ofFinite Element Methods*. Springer, 2016.

[56] A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[57] A. Logg and G. N. Wells. DOLFIN: automated finite element computing. *ACM Transactions on Mathematical Software*, 37, 2010.

[58] A. Logg, G. N. Wells, and J. Hake. DOLFIN: a C++/Python finite element library. In A. Logg, K.-A. Mardal, and G. N. Wells, editors, *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*, chapter 10. Springer, 2012.

[59] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[60] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[61] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[62] Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105, 1990.

[63] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[64] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[65] Konstantinos Agathos, Eleni Chatzi, and Stéphane P. A. Bordas. Multiple crack detection in 3d using a stable xfem and global optimization. *Computational mechanics*, 62:835–852, 2018.

[66] JC Grandidier and É Lainé. Identification by genetic algorithm of a constitutive law taking into account the effects of hydrostatic pressure and speeds. *Oil & Gas Science and Technology-Revue de l'IFP*, 61(6):781–787, 2006.

[67] Marvin Hardt, Deepak Jayaramaiah, and Thomas Bergs. On the application of the particle swarm optimization to the inverse determination of material model parameters for cutting simulations. *Modelling*, 2(1):129–148, 2021.

[68] Prakash Thakolkaran, Akshay Joshi, Yiwen Zheng, Moritz Flaschel, Laura De Lorenzis, and Siddhant Kumar. Nn-euclid: Deep-learning hyperelasticity without stress data. *Journal of the Mechanics and Physics of Solids*, 169:105076, 2022.

[69] Akira Isihara, Natsuki Hashitsume, and Masao Tatibana. Statistical theory of rubber-like elasticity. iv.(two-dimensional stretching). *The Journal of Chemical Physics*, 19(12):1508–1512, 1951.

[70] DW Haines and WD Wilson. Strain-energy density function for rubberlike materials. *Journal of the Mechanics and Physics of Solids*, 27(4):345–360, 1979.

[71] Alan N Gent and AG Thomas. Forms for the stored (strain) energy function for vulcanized rubber. *Journal of Polymer Science*, 28(118):625–628, 1958.

[72] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[73] SymPy Developers. Symengine, a fast symbolic manipulation library, written in c++, 2016.

A Representation of Mathematical Expressions

Mathematical expressions can be represented using graphs, which can then be used to perform downstream regression or classification tasks. Within a general approach, we can view mathematical operations, variables, or expressions, which we collectively call primitives, as graph node labels and define different expressions as different ways to connect these nodes. To explain this concept we first introduce some basic terminology on graphs, S-expressions and Polish notation. We then summarize the construction of mathematical expression using graph representations and their utilization in the context of symbolic regression.

Basic Glossary on Graphs: Throughout the manuscript, we consider nomenclature to describe different graphs and their nodes. For this purpose, we consider a short glossary accompanied by Figure 14 to explain these terms. A general graph is a topology whose edges are undirected, meaning that moving both from x to 3 and from 3 to x is allowed, all nodes can be connected with all other nodes without restrictions and also cycles, meaning a node is connected with itself, are allowed to exist. A directed acyclic graph is a topology where the edges are directed, meaning that moving only from x to 3 is allowed, all nodes can be connected with all nodes, and no cycles are allowed. A weighted directed acyclic is an directed acyclic graph whose edges have also weights. The weights can be thought of as the importance of edges or how strongly correlated two nodes are. A tree is a graph topology that nodes are connected to other nodes in a specific hierarchy. The node $-$ is called the parent of the nodes $\cdot$ and y which are called the children of $-$. A tree takes its name from the number of children that appear at most in it. For example, a tree where the nodes have two children is called binary, with three ternary and with k k-ary. The nodes that have no children are called leaf nodes and the node with no parent a root node in a tree, see the green and magenta nodes in Figure 14 respectively.

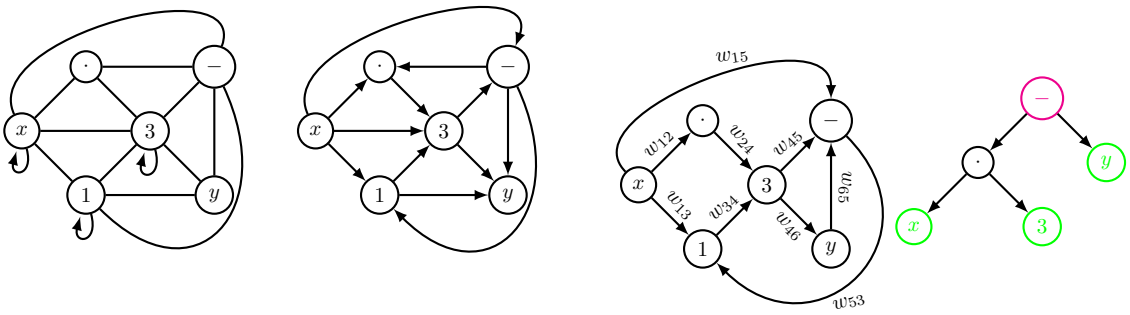
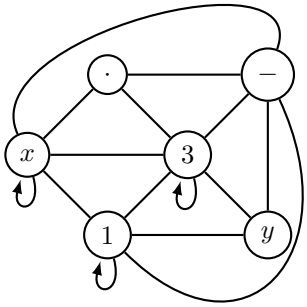


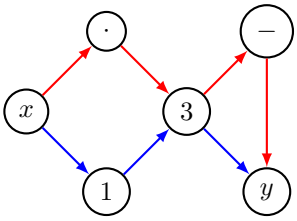
Figure 14: Examples of three graph topologies. On the far left, we have a general graph; In the middle left, a directed acyclic graph; In the middle right, a weighted directed acyclic graph; On the far right, a binary tree.

Graph and Tree Representations, S-expressions: A graph representation of a mathematical expression is one where the graph nodes represent primitives, see Figure 15 left, and the edges of the graph describe different possible orders of operations. By considering orders of operations between the primitives, implying a direction on the graph edges, and different paths on the graph, possible expressions are defined, see Figure 15 middle. A more simplified, yet also more restricted, representation that a priori assumes the order of the operations is the tree representation, see Figure 15 right. Any tree can be represented using a recursive structure of nested lists called symbolic or S-expressions.

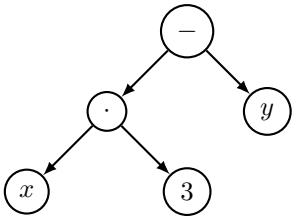
Polish Notation: To better clarify the relation between a mathematical formula and a tree, we introduce the so-called Polish notation. This is a mathematical notation where operators proceed operands. Mathematical operations on any number of arguments written in Polish notation can be easily translated into trees because their nested operation format is compatible with S-expressions. For example, let us consider the red path in Figure 15, which provides the expression $x \cdot 3 - y$. We can write this expression in Polish notation as $-(\cdot(x, 3), y)$, and then translate it into a tree structure, as shown in Figure 15 right.



An example of a general graph. Each node contains a primitive, i.e. either a mathematical operation or a variable.



The path in red provides the expression $x \cdot 3 - y$ or in Polish notation $-(\cdot(x, 3), y)$. The path in blue does not deliver a meaningful expression.



Tree representation of $-(\cdot(x, 3), y)$.

Figure 15: Left: a possible representation of mathematical expressions using an acyclic graph; middle: examples of paths on a directed acyclic graph; right: an example of a tree.

Construction of Expressions using Graph Representations: By choosing the possible order of operations between primitives, either by considering different paths in directed acyclic graphs or different trees, we can generate expressions. It is evident that not all paths provide meaningful mathematical expressions; e.g. in Figure 15 the blue path delivers $x13y$. To ensure that only meaningful expressions are generated, constraints can be considered in the choice of the graph paths or, in the case of trees, in the choice of the children given the parent. It is also clear that the space of possible expressions grows exponentially with the number of nodes; thus the choice of the primitives plays a crucial role on the size of this space.

Discovering Expressions from Data (Symbolic Regression): As discussed in the previous paragraphs, expressions are constructed by choosing orders of operations between nodes and paths that provide meaningful results. Symbolic regression is a systematic way of finding a possible path in a graph or a tree that corresponds to an expression that best fits given data. Different symbolic regressions algorithms differ by their choices of i. primitives, i.e. whether they are only mathematical operations or variables or whether they can also consist of sub-expressions, ii. graph representations, i.e. directed acyclic or tree representations, iii. schemes to enforce that the resulting expressions are meaningful. A further choice of that of the objective function to be minimized to optimize the fit of the given data, see Figure 16.

B Regular Tree Grammars

The purpose of this section is to provide a more detailed description of the properties of RTGs, and more specifically how production rules and trees/sub-trees are defined. Moreover, we present the parsing and generation algorithms with simple examples.

Properties of Regular Tree Grammars: A RTG is defined as the 4-tuple $\hat{\mathcal{G}} = \{\Phi, \tilde{\Sigma}, R, S\}$, where Φ a set of non-terminal symbols, $\tilde{\Sigma}$ a ranked alphabet, R a set of production rules, and S the starting non-terminal symbol. Consider $\tilde{\Sigma} = \{+ : 2, a : 0, 3 : 0\}$ for simplicity and $a + 3$ a derivation of the grammar using a list of production rules. We write $a + 3$ in Polish notation as $+(y, 3)$, then we can define a tree as $\hat{w} = s(\hat{t}_1, \hat{t}_2)$, where $x = +$, $\hat{t}_1 = a$, and $\hat{t}_2 = 3$. More generally, a tree with k children, a k-ary tree, can be defined as $\hat{w} = s(\hat{t}_1, ..., \hat{t}_k)$, where $s \in \tilde{\Sigma}$ an element of the alphabet, and $\hat{t}_1, ..., \hat{t}_k$ sub-trees defined using rules involving elements of the alphabet. In this definition $k = 0$ denotes leaf nodes, nodes without children, of the tree. The production rules are defined as $\Psi \to \hat{w}$ where $\Psi \in \Phi$ a non-terminal symbol and $\hat{w}$ a possible sub-tree derived by performing recursive substitutions of non-terminal symbols

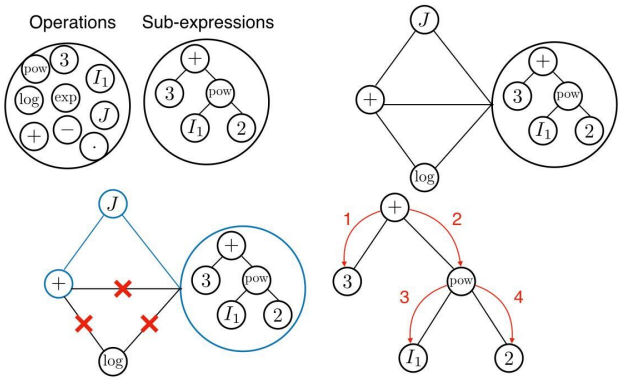
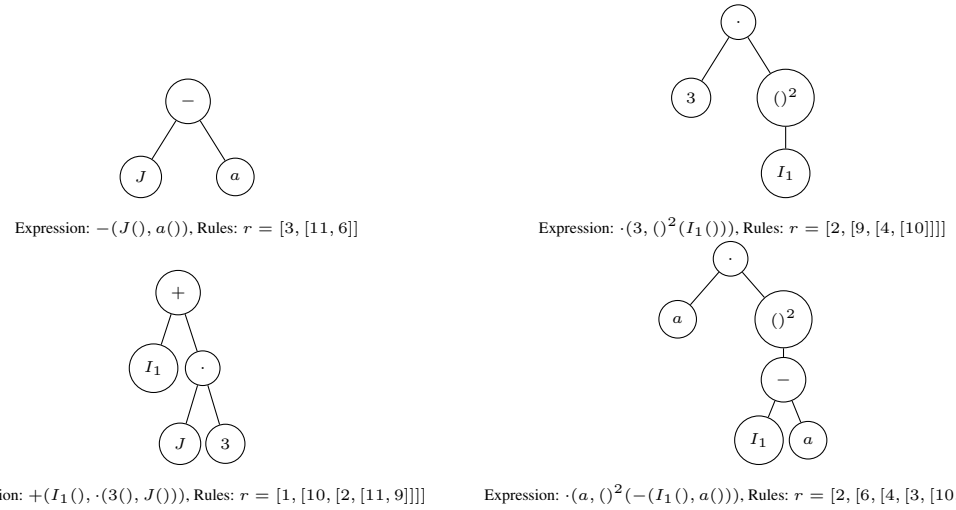


Figure 16: Symbolic regression methods can be constructed by choosing i. a set of primitives consisting of operations, and sub-expressions, top left, ii. a representation of the possible functions, i.e. directed acyclic graphs, top right, and iii. sampling valid expressions based on constraints.

starting from Ψ using production rules that have Ψ on their left-hand side. More specifically, the production rules for an operation s of arity k of the alphabet is defined $\Psi \rightarrow s(L_1, ..., L_k)$, where $L_1, ..., L_k \in \Phi$ non-terminal symbols. In Section 1 and 2, we discussed how the trees produced by grammar are characterized by production rules. In RTGs, the trees are derived in a recursive, hierarchical, manner and we represent this hierarchical structure of rules using nested lists. We provide examples of sub-tree expressions derived starting from symbol Ψ of $\hat{\mathcal{G}}_{NH}$, see Equation 20, together with the nested lists of production rules that derive them in Figure 17.



Expression: $-(J(), a())$, Rules: $r = [3, [11, 6]]$

Expression: $\cdot(3, ()^2(I_1()))$, Rules: $r = [2, [9, [4, [10]]]]$

Expression: $+(I_1(), \cdot(3(), J()))$, Rules: $r = [1, [10, [2, [11, 9]]]]$

Expression: $\cdot(a, ()^2(-(I_1(), a())))$, Rules: $r = [2, [6, [4, [3, [10, 6]]]]]$

Figure 17: Examples of sub-trees produced by the grammar. We present the derivations of the sub-trees in Polish notation to emphasize the fact that we are now working with tree grammars.

We are going to use the expressions in Figure 17 as examples, together with the grammar $\hat{\mathcal{G}}_{NH}$ presented in Equation 20, to explain how the nested list structures work. The list that contains the rules that generate the expression $-(J, a)$ is comprised by two levels. The second level $[11, 6]$ corresponds to the rules that generate the left and the right children of $-$, respectively. The first level $[3, ...]$ corresponds to the rule that generates $-$. Therefore, what the nested list $[3, [11, 6]]$ represents in words is "the first level of the tree is generated using the rule (3), and on the second level the left child is generated using rule (11) and the right is generated using rule (6). For more complicated cases, the same logic applied. For example in the expression $\cdot(a, ()^2(-(I_1(), a())))$, the nested list $[2, [6, [4, [3, [10, 6]]]]]$ can be connected to the tree structure as follows:

- The first level of the tree is derived by rule (2) which then created two children, meaning $r = [2, [,]]$.
- The left child in the second level is derived by rule (6), $r = [2, [6,]]$.The right child in the second level is created by first applying rule (4) which created one child, $r = [2, [6, [4, []]]]$.

- The child in the third level is derived by rule 3 and creates two children, $r = [2, [6, [4, [3, [,]]]]]$.
- The left child of the fourth level is derived by rule (10) and rule (6) is applied to derive the right child, $r = [2, [6, [4, [3, [10, 6]]]]]$.

$$
\begin{aligned}
&S = \{\ C\ \},\\
&\Phi = \{\ C, \Psi, L\},\\
&\tilde{\Sigma} = \{\ +:2, -:2, \cdot:2, ()^2:1, I_1:0, J:0,\ a:0\ , 1:0, b:0, 3:0\ \},\\
&R = \{\ C \rightarrow\ +(\Psi, \Psi), (\mathbf{1})\\
&\qquad \Psi \rightarrow \cdot(L, \Psi), (\mathbf{2})\\
&\qquad \Psi \rightarrow\ -(L, L), (\mathbf{3})\\
&\qquad \Psi \rightarrow\ ()^2(\Psi), (\mathbf{4})\\
&\qquad \Psi \rightarrow\ L, (\mathbf{5})\\
&\qquad L \rightarrow 0.5(), (\mathbf{6})\\
&\qquad L \rightarrow 1(), (\mathbf{7})\\
&\qquad L \rightarrow 1.5(), (\mathbf{8})\\
&\qquad L \rightarrow 3(), (\mathbf{9})\\
&\qquad L \rightarrow I_1(), (\mathbf{10})\\
&\qquad L \rightarrow J()(\mathbf{11})\ \}.
\end{aligned}
\tag{20}
$$

Each sub-list in the nested list structure describes one level of the tree hierarchy. Next we discuss the process of extracting this information given a tree and the process of generating a tree given a nested list structure in the general case. For implementing data-driven discovery of constitutive laws, RTGs need to be defined in a *deterministic manner*, meaning that two production rules cannot have the same right hand side. The reason that this property is favorable is that deterministic grammars are *unambiguous*, which means that each list of production rules uniquely generates one tree and vise versa [42]. In deterministic RTGs the parsing and generation procedures possess a $\mathcal{O}(|\hat{w}|)$ time, where $|\hat{w}|$ the size of the tree, and computation complexity [42].

Tree Parsing We discussed how expressions represented as trees can be characterized using grammar rules and provided some examples in Figure 17. In this section, we are going to discuss the process of extracting a characterization given a tree and a grammar, called parsing. Parsing takes a tree, i.e. $\hat{w} = +(I_1(), \cdot(3(), J()))$ as an input and provides a nested list of rules, i.e. $r = [1, [10, [2, [11, 9]]]]$ as an output. This is done by traversing the tree structure either starting from the bottom (bottom-up parsing) or starting from the top (top-down parsing), and extracting the rules that generate each level of the hierarchy. Consider a case where $\hat{w} = s(\hat{t}_1, ..., \hat{t}_k)$, and rules of the form $\Psi \rightarrow s(L_1, ..., L_k)$ that generate any possible tree $\hat{w}$ starting from Ψ. Bottom-up parsing finds the rule r_j that provides Ψ given L_j and top-down parsing tries to find the rule r_j that provide L_j given Ψ where $\Psi, L_j \in \Phi$ and $j \in [1, ..., k]$ denotes the indices of the children of a k-ary tree. Regular tree grammars consider a bottom-up parsing approach.

In a RTG, the parsing is done recursively starting from identifying the rule r_j with L_j on its right hand side that generated the sub-tree $\hat{t}_j$ with L_j as a starting point. For each step of the process the algorithm provides the rule r_j with L_j on its left-hand side and the non-terminal L_j for each $j \in [1, ..., k]$. Then R is searched for a rule with the non-terminal Ψ on its left-hand side, i.e. $r = \Psi \rightarrow s(L_1, ..., L_k)$, and returns a concatenation of the rules that generate each sub-tree, i.e. $[r, \hat{r}_1, ..., \hat{r}_k]$. If a rule that contains Ψ on the left hand is not found then the process fails, which means that a tree cannot be produced by the chosen grammar. If the recursive process successfully reaches the starting symbol S, then the sentence is in the language $\mathcal{L}(\mathcal{G})$. We provide the general form of the parsing algorithm in Algorithm 1.

Tree Generation Guided by a List of Rules: The inverse process of parsing is generation. The parsing process extracts a nested list of rules from a tree, while the generation constructs a tree based on a list of rules. Therefore, the generation algorithm takes as an input a list, i.e. $r = [1, 10, 2, 11, 9]$ and generates a tree, i.e. $\hat{w} = +(I_1(), \cdot(3(), J()))$, presented in Figure 17. This process does not require a nested list of rules, but instead a list of rules $r = [\hat{r}_1, ..., \hat{r}_{N_k}]$, where N_k the number of rules, and S the starting non-terminal. This process is performed by traversing the tree from the top to bottom and from left to right.

Consider a case where $\hat{w} = s(\hat{t}_1, ..., \hat{t}_k)$ a tree and $\Psi \rightarrow s(L_1, ..., L_k)$ the form of the grammar rules. The generation substitutes a non-terminal Ψ in a tree according to a rule $\Psi \rightarrow s(L_1, ..., L_k)$, and at the same time stores $L_k, ..., L_1$,

Algorithm 1 The tree parsing algorithm for a RTG $\hat{\mathcal{G}}$ considered in this work.

Input: An expression tree of the form $\hat{w} = s(\hat{t}_1, ..., \hat{t}_k)$.
Output: A nested list of rules $[r, \hat{r}_1, ..., \hat{r}_k]$ that generate $\hat{w}$.
function $parsing(\hat{w})$
  for $j = 1, ..., k$ **do**
    $L_j, \hat{r}_j = parsing(\hat{t}_j)$
  end for
  if $r = [\hat{r}_1, ..., \hat{r}_k]$ exists with $\hat{r}_1, ..., \hat{r}_k$ rules with $L_1, ..., L_k$ on their right-hand side such that $r : \Psi \rightarrow s(L_1, ..., L_k)$ **then**
    return Ψ, $[r, \hat{r}_1, ..., \hat{r}_k]$
  else
    There exists no rule sequence with the non-terminal Ψ on its right-hand side, therefore the expression is not in the language.
  end if

where $\Psi, L_j \in \Phi$ and $j \in [1, ..., k]$ denotes the indices of a k-ary tree. For each step of the process, the algorithm recovers the non-terminal L_j on the top of the list, checks if the given rule has L_j on the left-hand side, applies it and then removes L_j from the list. The process stops when the list of non-terminals is empty and all the pre-defined rules have been applied. If the list is not empty then the process fails.

Algorithm 2 The tree generation algorithm for a RTG $\hat{\mathcal{G}}$ and a list of rules $[r_1, ..., r_T]$ considered in this work.

Input: A list of rules $[\hat{r}_1, ..., \hat{r}_{N_r}]$ that generate $\hat{w}$.
Output: An expression tree of the form $\hat{w} = s(\hat{t}_1, ..., \hat{t}_k)$.
Create an empty list of non-terminals and store the starting symbol S.
function $generation(\hat{w})$
  for $j = 1, ..., N_r$ **do**
    Recover non-terminal Ψ from the tail of the list.
    Check if there a exists a non-terminal with Ψ on the left-hand side.
    Apply the rule $\Psi \rightarrow s(L_1, ...L_k)$.
    Replace Ψ in the generation with $s(L_1, ..., L_k)$
    Store $L_1, ..., L_k$ in the list.
  end for
  if The list in non-empty **then**
    The process fails.
  end if
return $\hat{w}$

An Example of Parsing and Generating for a Simple Tree: In this paragraph, we present a detailed example of the tree parsing and generation algorithms. We consider a RTG version of $\hat{\mathcal{G}}_{NH}$ and the expression $W(\mathbf{F}) = 3 \cdot I_1^2 + (J - 0.5)^2$ to explain the concepts of parsing and generation. We can write $W(\mathbf{F})$ in Polish notation as $W(\mathbf{F}) = +(\cdot(3(), ()^2(I_1())), ()^2(-(J(), 0.5())))$ and is be written as a tree:

$$\hat{w} = +(\hat{t}_1, \hat{t}_2),$$

where the sub-trees $\hat{t}_1$ and $\hat{t}_2$ are written as:

$$\hat{t}_1 = \cdot(\hat{t}_1^1, \hat{t}_1^2), \quad \hat{t}_2 = ()^2(\hat{t}_2^1),$$

the sub-trees $\hat{t}_1^1, \hat{t}_1^2$, and $\hat{t}_2^1$, as:

$$\hat{t}_1^1 = 3(), \quad \hat{t}_1^2 = ()^2(\hat{t}_1^{21}), \quad \hat{t}_2^1 = -(\hat{t}_1^{12}, \hat{t}_2^{12}),$$

and the sub-tree $\hat{t}_1^{21}, \hat{t}_1^{12}$ and $\hat{t}_2^{12}$ as:

$$\hat{t}_1^{21} = I_1(), \quad \hat{t}_1^{12} = J(), \quad \hat{t}_2^{12} = 0.5(),$$

Now we can apply the parsing algorithm 1 to $W(\mathbf{F})$, as follows:

- The algorithm begins from parsing $\hat{w} = +(\hat{t}_1, \hat{t}_2)$, which is composed from the sub-trees $\hat{t}_1$, and $\hat{t}_2$. The presence of the two sub-trees triggers recursion and the process considers parsing the sub-trees $\hat{t}_1 = \cdot(\hat{t}_1^1, \hat{t}_1^2)$ and $\hat{t}_2 = ()^2(\hat{t}_2^1)$ first.
- We consider the left sub-tree $\hat{t}_1 = \cdot(\hat{t}_1^1, \hat{t}_1^2)$ which is composed by $\hat{t}_1^1 = 3()$, and $\hat{t}_1^2 = ()^2(\hat{t}_1^{21})$. For the right sub-tree, the recursion is triggered and the algorithm parses the $\hat{t}_1^{21} = I_1()$.
- After parsing $\hat{t}_1^{21}$, which is in this case the bottom level, the algorithm continues to the upper levels of the hierarchy extracting the grammar rules that derive each level.
- The algorithm performs the same actions for the right sub-tree $\hat{t}_2 = ()^2(\hat{t}_2^1)$.

The application of the algorithm 1 can be written in detail as:

for $j = 1$ do
$L_1, \hat{r}_1 = parser(\hat{t}_1)$
  for $i = 1$ do
  $L_1^1, \hat{r}_1^1 = parser(\hat{t}_1^1)$
  return $L, 9$
  for $i = 2$ do
  $L_1^2, \hat{r}_1^2 = parser(\hat{t}_1^2)$
    for $l = 1$ do
    $L_1^{21}, \hat{r}_1^{21} = parser(\hat{t}_1^{21})$
      for $k = 1$ do
      $L_1^{21}, \hat{r}_1^{21} = parser(\hat{t}_1^{21})$
      return $L, 5$
    return $L, [5, [10]]$
  return $\Psi, [4, [5, [10]]]$
return $\Psi, [2, [9, [4, [10]]]]$
for $j = 2$ do
  for $i = 1$ do
  $L_2^1, \hat{r}_2^1 = parser(\hat{t}_2^1)$
    for $l = 1$ do
    $B_1^{12}, \hat{r}_1^{12} = parser(\hat{t}_1^{12})$
    return L, 11
    for $l = 2$ do
    $B_2^{12}, \hat{r}_2^{12} = parser(\hat{t}_2^{12})$
    return L, 6
  return $\Psi, [3, [12, 6]]$
return $\Psi, [4, [3, [12, 6]]]$
return $C, [1, [2, [9, [4, [5, [10]]]]]], [4, [3, [11, 6]]]]$

The final result of the algorithm is the nested list $r = [1, [2, [9, [4, [5, [10]]]]], [4, 3, [11, 6]]]]$ that characterizes the tree $\hat{w} = (\cdot(3(), ()^2(I_1())), ()^2(-(J(), 0.5())))$.

We now consider the inverse process for the previous example to showcase the generation algorithm. In this case, we provide the list of rules $r = [1, 2, 9, 4, 5, 10, 3, 11, 6]$ and the starting non-terminal S as an input and get the tree $\hat{w} = (\cdot(3(), ()^2(I_1())), ()^2(-(J(), 0.5())))$ as an output following Algorithm 2. The process is provided in detail:

Create an empty list of non-terminals and store the starting symbol S
  for $j = 1$ do
  Get S from the list.
  Apply rule $r_1 = S \to +(\Psi, \Psi)$.
  Replace Ψ with $+(\Psi, \Psi)$ in the tree generation.
  Now $\hat{w} = +(\Psi, \Psi)$.
  Store Ψ, Ψ in the list.

  for $j = 2$ do
  Get Ψ from the tail of the list.
  Apply rule $r_2 = \Psi \to \cdot(L, \Psi)$.

Replace Ψ with $\cdot(L, \Psi)$ in the tree generation.
Now $\hat{w} = +(\cdot(L, \Psi), \Psi)$.
Store L, Ψ in the list.

for $j = 3$ do
Get L from the tail of the list.
Apply rule $r_3 = L \to 3()$.
Replace L with $3()$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), \Psi), \Psi)$.
Rule r_3 did not produce a non-terminal to store.

for $j = 4$ do
Get Ψ from the tail of the list.
Apply rule $r_4 = \Psi \to ()^2(\Psi)$.
Replace Ψ with $()^2(\Psi)$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(\Psi)), \Psi)$.
Store Ψ in the list.

for $j = 5$ do
Get Ψ from the tail of the list.
Apply rule $r_4 = \Psi \to L$
Replace Ψ with L in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(L)), \Psi)$.
Store L in the list.

for $j = 6$ do
Get L from the tail of the list.
Apply rule $r_{10} = L \to I_1()$.
Replace L with $I_1()$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(I_1())), \Psi)$.
Rule r_{10} did not produce a non-terminal to store.

for $j = 7$ do
Get Ψ from the tail of the list.
Apply rule $r_4 = \Psi \to ()^2(\Psi)$.
Replace Ψ with $()^2(\Psi)$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(I_1())), ()^2(\Psi))$.
Store L, L in the list.

for $j = 8$ do
Get Ψ from the tail of the list.
Apply rule $r_3 = \Psi \to -(L, L)$.
Replace Ψ with $-(L, L)$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(I_1())), ()^2(-(L, L)))$.
Store L, L in the list.

for $j = 9$ do
Get L from the tail of the list.
Apply rule $r_{11} = L \to J()$.
Replace L with $J()$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(I_1())), ()^2(-(J(), L)))$.
Rule r_{11} did not produce a non-terminal to store.

for $j = 10$ do
Get L from the tail of the list.
Apply rule $r_6 = L \to 0.5()$.
Replace L with $0.5()$ in the tree generation.
Now $\hat{w} = +(\cdot(3(), ()^2(I_1())), ()^2(-(J(), 0.5()))))$.
Rule r_6 did not produce a non-terminal to store.

return $\hat{w} = +(\cdot(3(), ()^2(I_1())), ()^2(-(J(), 0.5())))$.

These two algorithms generalize for the general case of the list of rules $r = [\hat{r}_1, ..., \hat{r}_{N_r}]$ and $\hat{w} = s(L_1, ..., L_k)$ that generates $\hat{w}$.