

© 2019 Haizi Yu

AUTOMATIC CONCEPT LEARNING VIA INFORMATION LATTICES

BY

HAIZI YU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:

Assistant Professor Lav R. Varshney, Chair
Assistant Professor Ranjitha Kumar
Associate Professor Paris Smaragdis
Professor Igor Mineyev
Dr. Lester Mackey, Microsoft Research New England

Abstract

Concept learning is about distilling interpretable rules and concepts from data, a prelude to more advanced knowledge discovery and problem solving in creative domains such as art and science. While concept learning is pervasive in humans, current artificial intelligent (AI) systems are mostly good at either applying human-distilled rules (rule-based AI) or capturing patterns in a task-driven fashion (pattern recognition), but not at learning concepts in a human-interpretable way similar to human-induced rules and theory.

This thesis introduces a new learning problem—*Automatic Concept Learning (ACL)*—targeting *self-explanation* and *self-exploration* as the two principal pursuits; correspondingly, it proposes a new learning model—*Information Lattice Learning (ILL)*—combining *computational abstraction* and *probabilistic rule learning* as the two principal components. Woven around the core idea of abstraction, the entire ACL framework is presented as a generalization of Shannon’s information lattice that further brings learning into the picture. The core idea of abstraction is cast as a hierarchical, interpretable, data-free, and task-free clustering problem, seeded from universal priors such as simple algebra and symmetries.

The main body of the thesis comprises three self-contained yet close-knit parts: theory, algorithms, and applications. The theory part presents the mathematical *exposition* of ACL, formalizing the key notions of abstraction, concept, probabilistic rule, and further the entire concept learning problem. The goal is to lay down a solid path towards algorithmic means that are computationally feasible, reliable, and human-interpretable. The algorithms part presents the computational *development* of ACL, that is, ILL. It puts together computational abstraction and statistical learning in the same algorithmic picture, creating a bridge that connects deductive (rule-based) and inductive (data-driven) approaches in AI. Aiming for human interpretability and model transparency in particular, ILL in many ways mimics human learning. This includes mechanism-driven abstraction generation, as well as a “teacher-student” loop that can distill customizable traces of rules for data summarization and data explanation. The applications part *recapitulates* the theory and algorithms through concrete examples. Music is used for demonstration and automatic music concept learning is thoroughly studied. This part details the implementation of MUS-ROVER, an automatic music theorist that distills music composition rules from sheet music. To better support music ACL and music AI in general, the twin system MUS-NET is built as a crowdsourcing platform for making and serving digital sheet music data sets.

To my parents, for their love and support.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor Lav R. Varshney for the continuous support of my doctoral study and research, for his patience, motivation, and immense knowledge. He passed me the faith and courage to be brave—to be a brave, adventurous researcher exploring new problems/solutions in interdisciplinary boundaries between computer science and many other fields including mathematics, music, biology, etc. As a result, this thesis collects a mixture of computational, mathematical, and artistic ideas, and spans from theory to algorithms and to real applications, which would not be possible without his great support and guidance. Moreover, he has been a tireless advocate on my behalf, and always encouraged me not to be shy of showing my work to people from other institutions, national labs, industry, and connecting to people from all disciplines. The contributions that Lav has made to this thesis as well as to my general education and research character are very much appreciated.

I am grateful for the encouragement and suggestions provided by Ranjitha Kumar, Paris Smaragdis, Igor Mineyev, and Lester Mackey, the other members of my doctoral committee.

Research, at least from what I believe, is an interactive endeavor, especially considering the fact that the work done in this thesis comes from a diverse range of expertise. I would like to particularly thank all my research collaborators for their great contributions on my research publications: Guy Garnett, former faculty member from the university’s National Center for Supercomputing Applications (NCSA); Ranjitha Kumar, assistant professor from the department’s data-driven design group; Heinrich Taube, professor from the School of Music; Igor Mineyev, professor from the Department of Mathematics; Tianxi Li, assistant professor now from the Department of Statistics at the University of Virginia.

An important part of this work involves building MUS-ROVER and MUS-NET, the two music web applications that demonstrate the utilities of the theory and algorithms. These come from the great hands of two teams of undergraduate students that I have enjoyed working with. I would like to thank the whole MUS-ROVER team: Sarah Schieferstein, Fanbo Xiang, Dingcheng Yue, Benoit Ortalo-Magne, Daniela Zieba, Chen Pan, and the whole MUS-NET team: Seungjun Cho, Yirou Li, Carolyn Nye.

- Financial support provided in part by the IBM-Illinois Center for Cognitive Computing Systems Research (C3SR), a research collaboration as part of the IBM AI Horizons Network; and in part by grant number 2018-182794 from the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation.

Table of Contents

Chapter 1	Introduction	1
1.1	Automatic Concept Learning in a Nutshell	3
1.2	Information Lattice Learning in a Nutshell	7
1.3	Outline and Contributions	10
Chapter 2	Background	11
2.1	Math Preliminaries	11
2.2	Claude Shannon’s Information Lattice	13
2.3	Betty Shannon’s Stochastic Music and Thereafter	14
Chapter 3	Theoretical Exposition (Part I)	17
Chapter 4	Abstraction-Based Rules and Concepts	18
4.1	Abstraction: Literature Overview	18
4.2	Abstraction, Concept and Probabilistic Rule	20
Chapter 5	Computational Abstraction	22
5.1	Everyday Abstraction	22
5.2	Abstraction as Partition	24
5.3	Abstraction Hierarchy as Partition Lattice	25
5.4	Rule Hierarchy as Generalized Information Lattice	28
Chapter 6	Abstraction Generating Mechanisms	31
6.1	Feature-Induced Abstraction	31
6.2	Symmetry-Induced Abstraction	33
Chapter 7	Codetta: Summary and Discussions	41
Chapter 8	Algorithmic Development (Part II)	42
Chapter 9	Information Lattice Learning Phase I: Abstraction Generation	43
9.1	Feature Generation	43
9.2	Symmetry Generation: Top-Down Approach	45
9.3	Symmetry Generation: Bottom-up Approach	60
9.4	Restriction to Finite Subspaces	70
Chapter 10	Information Lattice Learning Phase II: Probabilistic Rule Learning	75
10.1	The Vanilla “Teacher \rightleftharpoons Student” Loop	75
10.2	The Adaptive Teacher: Rule Context-Matching	80
10.3	The Elastic Student: Rule Breaking	84

Chapter 11	Codetta: Summary and Discussions	96
Chapter 12	Applicational Recapitulation (Part III)	97
Chapter 13	MUS-ROVER: An Automatic Music Theorist and Pedagogue	99
13.1	Music Raw Representation	99
13.2	MUS-ROVER I	100
13.3	MUS-ROVER II	109
13.4	MUS-ROVER RB	121
13.5	MUS-ROVER Rules versus Music Theory	123
Chapter 14	MUS-NET: A Crowdsourced Home of Digital Sheet Music	132
14.1	Music AI and Music Big Data	132
14.2	Overview of Music Data Formats	135
14.3	Platform Principles	136
14.4	Platform Implementation	138
14.5	Music Transcription Contest	144
14.6	AI Applications in Action	147
Chapter 15	Coda	150
References	151

Chapter 1: Introduction

Is it possible to learn the laws of music theory directly from raw sheet music, in the same human-interpretable form as a music theory textbook? How little prior knowledge needs to be encoded in order to do so? This thesis considers these and similar questions in music and beyond, with an aim for developing a general framework for automatic concept learning.



Figure 1.1: Conceptual development in biological systems (top, green) and artificial systems (bottom, blue). Concept learning generally refers to the phase that happens before but leads to more efficient and robust problem-solving or task-performing.

In nature, the ability to distill rules and concepts from raw observations is pervasive in humans [1] and also in many other precocial species [2]. Further, it is evident that more advanced concepts can be developed once a “conceptual base” is established [1, 3, 4]. There are two directions one can pursue along this line of conceptual development (Figure 1.1: top): moving forward, more advanced concepts can yield further domain knowledge (e.g. laws of nature) as well as principled approaches to perform specific tasks [5]; moving backward, it still remains mysterious (or at least debatable) how a conceptual base is established in the first place, and is generally attributed to innate biology by nativists [3, 6–10].

Almost in parallel, we see similar situations in artificial intelligence (AI) which considers finding high-level representations (in terms of features, beliefs, rules, artificial neurons, etc.) of sensory data [11]. It is also evident that higher-level representations can be learned once a “prior” is established [12, 13]. Note that a prior can take many different forms in artificial intelligence and machine learning, e.g. atomic logic in automatic reasoning, distributions in Bayesian inference, visual descriptors in computer vision, or architectures in neural networks. There are also two directions one can pursue (Figure 1.1: bottom): moving forward, more advanced machine learning models can be devised from handcrafted priors to perform specific tasks automatically; moving backward, it is still considered a “black art” how a prior can be effectively designed in the first place, which is generally attributed to domain expertise or designers’ intuitions [14].

While the forward direction is widely studied both psychologically (in biological systems)

and algorithmically (in artificial systems), this thesis on automatic concept learning looks backward. We consider the general question of conceptualizing a data domain: a task-free preparation phase before any specific problem-solving [15]. To do this, we consider priors that encode very little domain knowledge (i.e. close to an innate conceptual base) but lead to abundant, structured, and human-interpretable domain concepts. In particular, we want to ask the question: how far back can we push a prior along the axis in Figure 1.2, so that it is decoupled from existing domain knowledge as much as possible while still being able to recover and even to complete what we know? The ultimate goal of automatic concept learning is to learn domain concepts/knowledge from universal priors—priors that encode no domain knowledge.

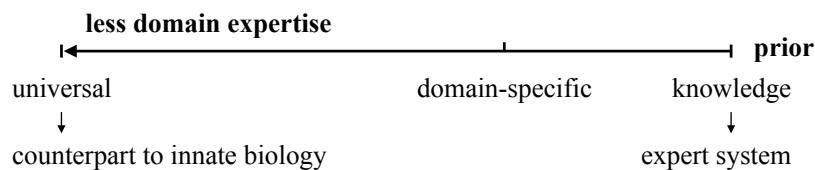


Figure 1.2: The pushback of a prior: from hardcoded domain knowledge to universal priors.

Although automatic concept learning in this thesis shares many commonalities with representation learning in deep learning [11], knowledge discovery in databases [16], as well as pattern recognition in machine learning [17] especially considering its role in summarizing data, it is distinct in at least two aspects. The first distinction is reminiscent of the difference between pattern recognition and pattern theory [18] as Ulf Grenander put it [19]: “it (pattern theory) differs from pattern recognition in that the latter emphasizes the construction of recognition algorithms whereas in pattern theory the representation of subject-matter knowledge is the centre of attention.” The second distinction is even more crucial: *interpretability* plays a central role in automatic concept learning and is the main theme throughout this entire thesis. Here, we require not only the learned results but also the entire leaning process to be interpretable. More precisely, the learning process and the learned results must be both machine-readable and human-interpretable.

As a result, we position automatic concept learning discussed in this thesis as a first step towards a principled machinery that learns and thinks cognitively like people—that “starts like a baby and learns like a child” [5, 20]. The main originality of this thesis is twofold: 1) to formalize a new learning problem, called *Automatic Concept Learning (ACL)*; 2) to propose a new learning model, called *Information Lattice Learning (ILL)*.

1.1 AUTOMATIC CONCEPT LEARNING IN A NUTSHELL

We first present an overview of automatic concept learning (ACL) without delving too much into its technical details which will be elaborated in later chapters. In this overview, we first give an informal description of ACL in terms of its input and output as well as its goals and motivations, then summarize the uniqueness of ACL compared to other classic problems and models in AI and machine learning. We conclude by describing the need to formalize ACL as a new learning problem that further needs a new learning model to solve.

1.1.1 What is ACL (Input and Output)?

Informally speaking, *automatic concept learning* is an algorithmic process of learning experiential rules (empirical laws) from sensory data (observations), or more broadly the process of distilling theory from phenomenology. As such, an ACL problem takes as input a data set, and outputs rules to summarize and to explain concepts that underlie the input data. It is important to recognize that the input-output direction here is opposite to that of a rule-based AI, exactly the same distinction between a music theorist and a composer (Figure 1.3).

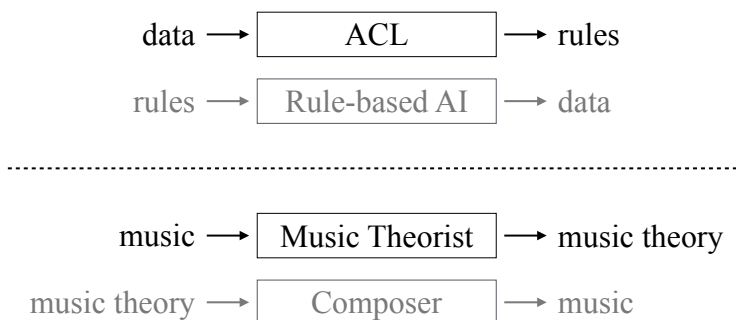


Figure 1.3: The input-output distinction between a concept learner and a rule-based AI (top), which is analogous to the distinction between a music theorist and a composer (bottom).

Let me name a few ACL examples. From a data set of right triangles in a 2D plane, we expect an output that rediscovers the Pythagorean theorem. From a data set of Copernicus' astronomical measurements, we expect an output that rediscovers heliocentric laws of the solar system [21]. From a data set of single cell RNA sequences from the retina, we expect an output that reveals the laws of pattern formation in neurodevelopment [22]. Lastly, taking a music data set of Bach's chorales (the main ACL application in this thesis), we expect an output that explains what makes Bach's chorales Bach's chorales and if possible, that further teaches us to write chorales in the style of Bach.

While the input is the same as in many data-driven learning problems, an ACL output differs from many classical learning settings such as traditional supervised and unsupervised learning [17]. In particular, an output consisting of predictions of class labels does not serve the purpose for concept learning; an output consisting of data samples that “counterfeit” the input data does not serve the purpose for concept learning either. More concretely, thinking about a music data set consisting of Bach’s Chorales as an example, a concept learner does not output labels to predict whether a new piece is written by Bach or others, nor does it output music that sounds like Bach’s chorales, but instead it outputs *ways of making music*, also known as music metacreation—the creation of a creator to create.

1.1.2 Why Automatic Concept Learning (New Goals/Motivations)?

The motivations, and thus also the goals, for automatic concept learning are twofold: to *explore* and to *explain* what we see (raw data or its raw distributions). Therefore, we make self-exploration and self-explanation the two hallmarks of automatic concept learning.

On the one hand, being *self-exploratory* enables a machine to assist people in research (e.g. to help the medical imaging diagnosticians by carrying out some of the time consuming discoveries while leaving the final decision to human judgment) and to inspire people to be more creative (e.g. to suggest new musical findings and possibilities to composers while still leaving the final composition to human judgment). These further target augmented intelligence and augmented creativity, for more efficient knowledge discovery and more personalized knowledge application.

On the other hand, being *self-explanatory* allows people to better understand the hidden laws in nature (e.g. to see what makes a cancer cell, cancer), to better understand other artificial models (e.g. to interpret a black-box algorithm), and even to better understand our own mental models (e.g. to shed some light on what the baby boy Mozart was thinking when coming up with his first minuet at the age of four). Such better understanding is important for AI safety, for more human-like generalization (within a domain), and for more human-like transfer learning (across domains).

It is worth noting that the above two goals/motivations are really two sides of the same coin: self-exploration means that we do not want a machine to blindly follow what people say; whereas self-explanation means that we do not want people to blindly follow what a machine says but instead we want people to know why (e.g. why AlphaGo made that crucial move that eventually led to victory). To embody the above high-level ideas of self-exploration and self-explanation, we next informally discuss the desired properties of an ACL output and the desired properties of an ACL model, whose formal presentations will be in later chapters.

1.1.3 Overview: Desired ACL Output

We expect an ACL output to comprise rules (Figure 1.3), each of which (hopefully from a different perspective) summarizes and explains some insight of the input data or data distribution. Instead of directly giving a formal definition of the rules, we first list a few desired properties of an ACL output, which eventually leads to its final mathematical form.

- **Typical.** The output must summarize the data well and reveal insights: it must capture typical regularities in the data rather than random patterns. This is later achieved by optimizing information-theoretic functionals for typicality and regularity.
- **Structured.** The output must be structured rather than being just a plain rule set. In particular, we express rule structure from both a static and a dynamic perspective:
 - **Hierarchical (static).** The output must be organized to show internal relations among its compositional rules, as concepts can be relatively more specific/general and rules can be relatively stronger/weaker. This is later achieved by superimposing a lattice structure on the rule family to encode a rule hierarchy.
 - **Sequential (dynamic).** The output must reflect a complete learning procedure rather than a set of unordered rules, since concept learning, by definition, is a process rather than an end result. Thus, rules are ordered in a sequence, by each time learning a new, independent rule that is not implied by previously learned rules. This is later achieved by eliciting a rule trace from a self-learning loop. Note that multiple traces are possible for the same input data (i.e. multiple valid ways to summarize and to explain the same data), leaving room for personalization.
- **Interpretable.** The output must be both machine readable and human-interpretable¹. We express this from both an individual (shallow) and a collective (deep) perspective:
 - **Individually interpretable (shallow).** The output must consist of rules that are individually interpretable rather than just a bunch of numerals. This is later achieved by explicitly representing every single rule as a statistical pattern on a mechanism-driven abstraction/clustering (rather than a data-driven clustering like k -means), and the presence of a clear mechanism assures explainability [25].
 - **Collectively interpretable (deep).** The output must consist of rules that are collectively interpretable, so that not only the rules but also the entire rule hierarchy is comprehensible. This is later achieved by a white-box algorithm that explicitly constructs the rule hierarchy.

¹Interpretability here is asserted axiomatically, rather than with explicit human experiments [23, 24].

1.1.4 Overview: Desired ACL Model

In parallel with the two goals for automatic concept learning (i.e. to explore and to explain), we list the following two desired properties of an ACL model.

- **Self-exploratory.** We start with some prior (i.e. a conceptual base) as a set of building blocks which can systematically span a concept universe for an automatic concept learner to self-explore. A desired prior consists of very little domain knowledge, ideally being a universal prior. This is in contrast with expert systems (e.g. a rule-based AI) which, from the perspective of concept learning, are considered “cheating” if one hard encodes all the known rules in a program [26]. Instead, we gradually strip domain knowledge away from a prior by a two-step attempt, where we first consider feature-induced concepts and then a more general setup—symmetry-induced concepts—to push the prior further back toward a universal one that emulates innate biology (Figure 1.2). When experimenting with specific domains, we further want the prior to span a concept universe that is expressive enough to cover much of the existing domain concepts, ideally a “closure” that completes what we knew and suggests what we might have overlooked. Lastly, we also want an efficient exploration algorithm that can traverse the spanned concept universe in an intelligent way.
- **Self-explanatory.** Beyond the two-level interpretability (shallow versus deep) of an ACL output mentioned in Section 1.1.3, we want the entire ACL model to be self-explanatory, i.e. to be a transparent white-box model. This is later achieved by a “teacher \rightleftharpoons student” learning loop that mimics a typical human learning scenario where a teacher gradually guides a student to a designated goal through iterative exercises and refinements. This learning architecture is similar to a generative adversarial network [27], and indeed the teacher is a discriminative model and the student is a generative model. However, what makes a key difference here is that both the discriminator (teacher) and the generator (student) are transparent to each other (e.g. allowing parameter sharing if in a generative adversarial network), and also transparent to us. In other words, the entire learning process is transparent to both insiders and outsiders. This further allows the model to work in both a human-out-of-the-loop mode and a human-in-the-loop mode, where the latter can further yield personalized education when replacing the automatic student with a human student.

Combining the above two properties leads to our information lattice learning, first briefly introduced in the following subsection.

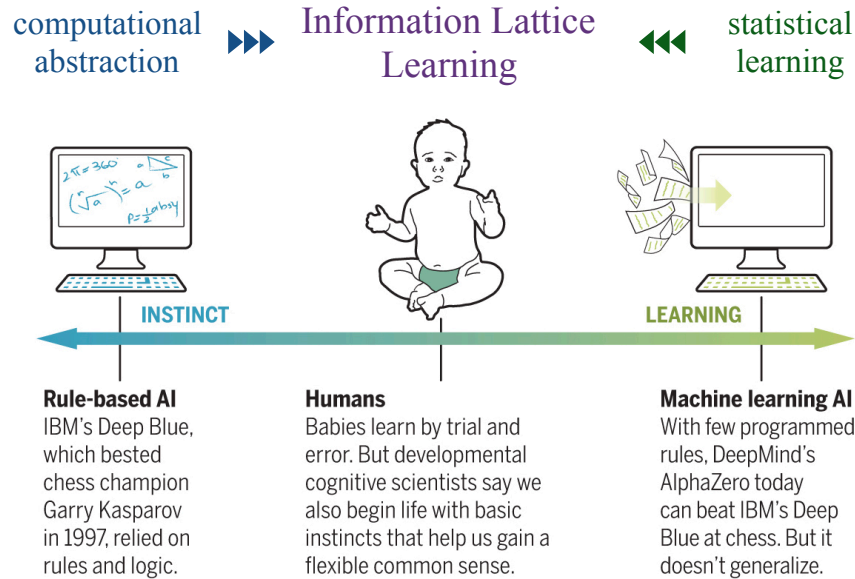


Figure 1.4: Information Lattice Learning (ILL), bringing together rule-based AI and machine learning AI, incorporates computational abstraction and statistical learning as its two core components. (Bottom figure by courtesy of N. Desai/Science [28].)

1.2 INFORMATION LATTICE LEARNING IN A NUTSHELL

To achieve the new goals brought about by automatic concept learning (Section 1.1.2), we need a new learning model, one that can ideally bring together rule-based AI and machine learning AI to achieve self-exploration and self-explanation at the same time. This, according to many developmental cognitive scientists, indeed mimics how a human develops—that starts like a baby and learns like a child [5, 20]. This further gives birth to our new learning model—Information Lattice Learning (ILL)—consisting of *computational abstraction* and *statistical learning* as its two core components (Figure 1.4).

1.2.1 Computational Abstraction

The cornerstone to define rules and concepts in automatic concept learning is *abstraction*, which is itself defined by a partition of—essentially a clustering problem for, or an equivalence relation on—a data space. The intuition here is that by partitioning (clustering or an equivalence relation), we obtain a coarse-grained view of the data space, where we identify every partition cell (cluster or equivalence class) as a whole and abstract away within-cluster variations. The key idea is to *forget* (low-level details) and to focus one's *attention* (on high-level concepts). As a result, the partition representation echoes the nature of abstraction,

which treats some distinct instances as if they were the same [29], or to be precise, equivalent, e.g. calling dogs and cats indistinguishably mammals.

Given an abstraction of a set represented by a partition (of that set), every cell in the partition—a cluster or an equivalence class—defines a *concept*, and the pair consisting of the abstraction and a probability distribution over the abstracted concepts defines a *rule*. Since a rule is defined as some probabilistic pattern of an abstraction, it is more precisely called a *probabilistic rule*, which specifies the form of an ACL output. Note that for any probabilistic rule defined as above, we can readily unravel its contained information by saying “we have discovered something probabilistically interesting if we abstract data in this way,” which attains the individual (shallow) interpretability of a desired ACL output (Section 1.1.3).

Beyond its intuitive sense, abstraction by partition naturally extends to abstraction hierarchy by partition lattice, from which the hierarchies of rules and concepts are immediate and also rigorously defined. By following this path, we will see in later chapters that the resulting rule hierarchy generalizes Shannon’s information lattice [30], which attains the collective (deep) interpretability of a desired ACL output (Section 1.1.3).

Moving forward from the above mathematical formalism to practical ways of computing abstraction-based rules and hierarchy, we develop computational abstractions via abstraction generating functions. An abstraction generating function is a formal way of mapping some generating source to its generated abstractions (in terms of a partition lattice). This thesis presents two types of generating sources, yielding feature-induced and symmetry-induced abstractions, respectively. A *feature-induced abstraction* is a partition whose cells are preimages of every individual feature value; treating features as quotient maps, we can further study the *topological* structure of abstraction spaces. A *symmetry-induced* abstraction is a partition whose cells are orbits that are invariant under the symmetry; representing symmetries by groups, we can further study the *algebraic* structure of abstraction spaces, as well as a *primal-dual* perspective of subgroup lattices and partition lattices. Both types of generating sources can be systematically assembled from basic building blocks (i.e. priors): feature functions are composed by basis descriptors; groups are generated by generators.

Crucially, our mathematical formalism and computational construction of abstraction, rule and concept, as well as their corresponding hierarchies, give rise to an implementation of a fully automated cycle of discovery, solving a long-standing challenge in building discovery systems for “concept formation” [31]. The cycle of discovery in our case, called a self-learning loop, further brings statistical learning into information lattices, which will be briefly introduced in the next subsection.

1.2.2 Statistical Learning in Shannon’s Information Lattice

Information lattice learning (ILL) occurs when computational abstraction meets statistical learning. ILL is not just about learning raw statistics from data, but also learning the right set of abstractions and their high-level statistics. We enable automatic concept learning by bringing statistical learning into information lattices. This is realized by a pair of bilateral subprocesses, namely *rule abstraction* and *rule realization*, also known as *reduction* and *elaboration* in some topic domains [32]. More specifically, the former is a statistical inference problem, projecting low-level data distributions onto high-level abstraction spaces; the latter is an inverse inference problem, or an inverse projection problem, resulting in a *higher level of generalization*: besides generating unseen data that satisfies the data distribution (generalization in the traditional sense), we generalize the data distribution itself as long as the rules—statistical patterns of the abstractions—are satisfied. This higher-level generalization further translates to the following advancement:

$$\text{memorizing data} \rightarrow \text{memorizing data distribution} \rightarrow \text{memorizing rules.} \quad (1.1)$$

Both subprocesses are formalized as a sequence of optimization problems whose formulations evolve in a systematic way. This systematic formalism eventually makes it possible to build a fully automated self-learning loop—the core learning algorithm—where no human intervention is needed during the entire course of concept learning. As stated earlier as one desired property of an ACL model (Section 1.1.4), the self-learning loop adopts a “teacher \rightleftharpoons student” architecture with the main idea being learning by comparison. In particular, the teacher, as a discriminative model, performs rule abstraction; whereas the student, as a generative model, performs rule realization. As such, the two subprocesses are executed by the teacher and the student in an alternating and iterative fashion. This “teacher \rightleftharpoons student” learning captures a typical human pedagogical scenario where a teacher guides a student to designated goal through iterative exercises and feedback [33].

Note that the loop operates on our generalization of Shannon’s information lattice, which itself encodes a hierarchy of abstractions and rules and is algorithmically constructed from group-theoretic foundations. In particular, we can further show that learning this hierarchy of invariant concepts essentially involves iterative optimization of Bayesian surprise and entropy, which further couples group theory and information theory during learning.

ILL gives a first step towards a principled and a cognitive ACL that can further lead to subsequent knowledge discovery. Later in the Applications Part of this thesis, we will detail the implementation of practical automatic concept learners, also called automatic theorists, for specific topic domains.

1.3 OUTLINE AND CONTRIBUTIONS

The main contributions of the thesis are separated into three self-contained yet close-knit parts, cohesively organized into a complete sonata form (Figure 1.5).

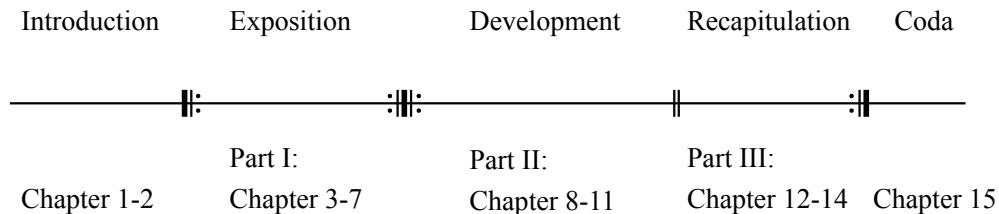


Figure 1.5: Thesis outline in a sonata form.

Part I presents the mathematical exposition of ACL, formalizing the key notions of abstraction, concept, probabilistic rule, and further the entire ACL problem (Chapter 4). The whole theory is grounded on a mathematical construct of abstraction, which is then systematically derived towards abstraction hierarchy (Chapter 5) and its generating mechanism (Chapter 6). Note that the goal in mathematizing the notion of abstraction is not for its own sake, but to go from this theoretical base towards computational developments that further bring about feasible, reliable, and human-interpretable algorithmic models.

Part II presents the algorithmic development of ACL, proposing the new learning model ILL. ILL includes two phases, with the first phase—abstraction generation (Chapter 9)—being the deductive preparation phase and the second phase—probabilistic rule learning (Chapter 10)—being the subsequent inductive learning phase. The two phases are the constituent pillars of ILL, which couples abstraction with statistics to fill in the gap between a rule-based AI and a machine learning AI. The resulting ILL model achieves a deeper-level interpretability, wherein not only the learned results (i.e. rules and concepts) are interpretable, but also the entire leaning process (i.e. the model itself) is comprehensible to people.

Part III recapitulates Parts I and II through real-world applications. We start with music as the first application domain and thoroughly discuss automatic music concept learning. This part presents a detailed implementation of the three versions of MUS-ROVER, an automatic music theorist that distills, from sheet music, music composition rules (Chapter 13). MUS-ROVER is further wrapped into a web application and serves as an automatic music pedagogue that delivers the rules as personalized music composition lessons. To better support music ACL and music AI in general, the twin system MUS-NET is built as an online crowdsourcing platform for making and serving digital sheet music data sets (Chapter 14).

Chapter 2: Background

The theory of Automatic Concept Learning and Information Lattice Learning is directly built from abstract mathematics. The resulting machinery, all constructive and operational, provides theoretical support for both the feasibility and the efficiency of the algorithms proposed thereafter. It is also the foundation on which both the generality of ACL and the interpretability of ILL are grounded. Despite its independent development, part of the theory is later shown as a generalization of Claude Shannon’s information lattice, from which we borrow the name for part of our learning model. As a result, this background chapter first presents a minimal core of math preliminaries (for review and notation purposes only) as well as Shannon’s information lattice in its original context.

The main application discussed in this thesis is in the domain of music, but from the computational perspective of music composition and music intelligence yielded thereafter. As a result, this chapter further presents historical background on computer music, starting from Betty Shannon (Claude’s wife) and her music composition via stochastic processes.

2.1 MATH PRELIMINARIES

Our intent in this section is twofold: to set up the theoretical playground for our own theoretical and algorithmic work; and to lay down the technical terms and notations that we use in this thesis. There is no contribution of our own in this section. We will be brief, referring readers to standard textbooks [34–37] for details.

2.1.1 Partition and Equivalence Relation

A *partition* \mathcal{P} of a set X is a collection of mutually disjoint, non-empty subsets of X whose union is X . Elements in \mathcal{P} are called *cells* (or less formally, *clusters*); the size of \mathcal{P} is $|\mathcal{P}|$, i.e. the number of cells in \mathcal{P} . An *equivalence relation* on a set X , denoted \sim , is a binary relation satisfying reflexivity, symmetry, and transitivity. An equivalence relation \sim on X induces a partition of X : $\mathcal{P} = X/\sim := \{[x] \mid x \in X\}$, where the quotient X/\sim is the set of *equivalence classes* $[x] := \{x' \in X \mid x \sim x'\}$. Conversely, a partition \mathcal{P} of X also induces an equivalence relation \sim on X : $x \sim x'$ if and only if x, x' are in the same cell in \mathcal{P} .

2.1.2 Lattice (Partial Order)

A *partial order* is a binary relation that satisfies reflexivity, antisymmetry, and transitivity. A *lattice* is a partially ordered set (or a *poset*) in which every pair of elements has a unique supremum (i.e. least upper bound) called the *join* and a unique infimum (i.e. greatest lower bound) called the *meet*. For any pair of elements p, q in a lattice, we denote their join and meet by $p \vee q$ and $p \wedge q$, respectively. A *sublattice* is a nonempty subset of a lattice, which is closed under join and meet. A *join-semilattice* (resp. *meet-semilattice*) is a poset in which every pair of elements has a join (resp. meet). So, a lattice is both a join-semilattice and a meet-semilattice. A lattice is *bounded* if it has a greatest element and a least element.

2.1.3 Group Theory

Group and subgroup. A *group* is a pair $(G, *)$ where G is a set and $*$: $G \times G \rightarrow G$ is a binary operation satisfying the *group axioms*: associativity, the existence of identity (denoted e), and the existence of inverse. We also directly say that G is a group, whenever the group operation is understood. Given a group $(G, *)$, a subset $H \subseteq G$ is a *subgroup*, denoted $H \leq G$, if $(H, *)$ is a group. The singleton $\{e\}$ is a subgroup of any group, called the *trivial group*.

Subgroup lattice. Let G be a group. We use \mathcal{H}_G^* to denote the collection of all subgroups of G . The binary relation “a subgroup of” on \mathcal{H}_G^* , denoted \leq , is a partial order. (\mathcal{H}_G^*, \leq) is a lattice, called the *lattice of all subgroups* of G , or the *(complete) subgroup lattice* for G in short. For any pair of subgroups $A, B \in \mathcal{H}_G^*$, the join $A \vee B = \langle A \cup B \rangle$ is the smallest subgroup containing A and B ; the meet $A \wedge B = A \cap B$ is the largest subgroup contained in A and B .

Generating set of a group. Given a group G and a subset $S \subseteq G$, the subgroup (of G) generated by S , denoted $\langle S \rangle$, is the smallest subgroup of G containing S ; equivalently, $\langle S \rangle$ is the set of all finite products of elements in $S \cup S^{-1}$ where $S^{-1} := \{s^{-1} \mid s \in S\}$. The subgroup generated by a singleton $S = \{s\}$ is called a *cyclic group*; for brevity, we also call it the subgroup generated by s (an element), denoted $\langle s \rangle$.

Group action. Let G be a group and X be a set, then a *group action* of G on X (or G -action on X) is a function $\cdot : G \times X \rightarrow X$ that satisfies identity ($e \cdot x = x, \forall x \in X$) and compatibility ($g \cdot (h \cdot x) = (g * h) \cdot x, \forall g, h \in G, \forall x \in X$). In this thesis, we adopt by default the multiplicative notation for group operations and actions, in which \cdot or $*$ or both may be omitted. For any G -action on X , the *orbit* of x under G is the set $Gx := \{g \cdot x \mid g \in G\}$, and the *quotient* of X by G -action is the set consisting of all orbits $X/G := \{Gx \mid x \in X\}$. For

any G -action on X , the (*pointwise*) *stabilizer* of $x \in X$ is the set $G_x := \{g \in G \mid g \cdot x = x\} \leq G$; the *setwise stabilizer* of $Y \subseteq X$ is the set $G_Y := \{g \in G \mid g \cdot Y := \{g \cdot x \mid x \in Y\} = Y\} \leq G$.

Group conjugacy. Let G be a group. We say that two elements $a, b \in G$ are *conjugate* to each other, if there exists a $g \in G$ such that $b = gag^{-1}$, and two subsets $A, B \subseteq G$ are *conjugate* to each other, if there exists a $g \in G$ such that $B = gAg^{-1}$. In either case, conjugacy is an equivalence relation on G (resp. 2^G , i.e. the power set of G), where the equivalence class of $a \in G$ (resp. $A \subseteq G$) is called the *conjugacy class* of a (resp. A). In particular, we can restrict the above equivalence relation on 2^G to the collection of all subgroups \mathcal{H}_G^* which is a subset of 2^G .

Group homomorphism. Let $(G, *)$ and (H, \cdot) be two groups. A function $\phi : G \rightarrow H$ is called a *homomorphism* if $\phi(a * b) = \phi(a) \cdot \phi(b)$ for all $a, b \in G$. An *isomorphism* is a bijective homomorphism. We say two groups G and H are *homomorphic* if there exists a homomorphism $\phi : G \rightarrow H$, and say they are *isomorphic*, denoted $G \cong H$, if there exists an isomorphism $\phi : G \rightarrow H$.

Group decomposition. Let S be a subset of a group G ; then $N_G(S) := \{g \in G \mid gSg^{-1} = S\}$ is called the *normalizer* of S in G , which is a subgroup of G . We say a subset $T \subseteq G$ *normalizes* another subset $S \subseteq G$ if $T \subseteq N_G(S)$. We say a subgroup N of a group G is a *normal subgroup* of G , denoted $N \trianglelefteq G$, if G normalizes N , i.e. $G = N_G(N)$. Let G be a group, $N \trianglelefteq G$, $H \leq G$, $N \cap H = \{e\}$, and $G = NH$; then NH is the *inner semi-direct product* of N and H , and $N \rtimes H$ is the *outer semi-direct product* of N and H . The outer semi-direct product $N \rtimes H$ is the group of all ordered pairs $(n, h) \in N \times H$ with group operation defined by $(n, h)(n', h') = (nhn'h^{-1}, hh')$. The inner and outer semi-direct products are isomorphic, i.e. $NH \cong N \rtimes H$. The semi-direct product equation $G = NH$ gives a decomposition of G into “nearly non-overlapping” (i.e. with trivial intersection) subgroups; moreover, for any $g \in G$, there exist a unique $n \in N$ and $h \in H$ such that $g = nh$.

2.2 CLAUDE SHANNON’S INFORMATION LATTICE

In his 1953 work, Claude E. Shannon attempted to unravel the nature of information [30] (beyond just quantifying its amount which is mutual information [38]). In the specific context of communication problems, he first coined the term *information element* to denote the nature of information which is invariant under “(language) translations” or different encoding-decoding schemes. He further introduced a partial order between a pair of information elements, eventually yielding a lattice of information elements, or *information lattice* in short. We present an overview of Shannon’s original work and a follow-up work [39] which formalizes Shannon’s idea in a more principled way.

Information element. An *information element* is an equivalence class of random variables (of a common sample space) with respect to the “being-informationally-equivalent” relation, where two random variables are informationally equivalent if they induce the same σ -algebra (of the sample space). Under this definition, the notion of an information element—essentially a probability space—is more abstract than that of a random variable: an information element can be realized by different random variables. The relationship between different but informationally-equivalent random variables and their corresponding information element is analogous to the relationship between different translations (say, English, French, or a code) of a message and the actual content of the message. Since different but faithful translations are viewed as different ways of describing the same information, the information itself is then regarded as the equivalence class of all translations or ways of describing the same information. Therefore, the notion of information element reveals the nature of information.

Information lattice. An *information lattice* is a lattice of information elements, where the partial order is defined by $x \leq y \iff H(x|y) = 0$ where H denotes the Shannon entropy. The join of two information elements $x \vee y = x + y$ is called the total information of x and y ; the meet of two information elements $x \wedge y = xy$ is called the common information of x and y . By definition, under a fixed probability measure, every information element can be uniquely determined by its induced σ -algebra. Also, it is known that every σ -algebra of a countable sample space can be uniquely determined by its generating (via union operation) sample-space-partition. Thus, an information lattice has a one-to-one correspondence to a partition lattice. Further, given a partition of a sample space, one can construct a unique permutation subgroup whose group action on the sample space produces orbits that coincide with the given partition [39]. Therefore, under this specific construction, any partition lattice has a one-to-one correspondence to the constructed subgroup lattice [39]. The above thought process can be succinctly summarized in the following chain:

$$\text{information lattice} \rightarrow \text{partition lattice} \rightarrow \text{subgroup lattice} \ (\rightarrow \text{interpretation}). \quad (2.1)$$

which further achieves group-theoretic interpretations of various information-theoretic results, bringing together information theory and group theory [40, 41].

2.3 BETTY SHANNON’S STOCHASTIC MUSIC AND THEREAFTER

With the passing of Mary Elizabeth Moore “Betty” Shannon on May 1, 2017, there has been renewed interest in her Bell Labs work at the intersection of music and mathematics

(mutual loves she shared with her husband, Claude [42]), which fits into the main theme of this thesis at the intersection of computer science, mathematics, and music. In this section, we first discuss some context around her work described in the technical memorandum “Composing Music by a Stochastic Process” (1949), together with the intellectual arc of computational creativity for music composition that has carried forward to the present.

Betty Shannon studied mathematics at the New Jersey College for Women (now Douglass College at Rutgers University) on a full scholarship and graduated Phi Beta Kappa. The next day, she started work at Bell Laboratories in Manhattan as a computer in the mathematics department; she was later promoted to technical assistant [42, 43]. For a 200-year period in the history of science and technology, human workers called “computers” were employed to perform calculations by hand, whether in computing the 1758 return of Halley’s comet, mathematical tables during the Great Depression, or ballistics tables during WWII. Sometime during the war, Warren Weaver started calling computers “girls” and one member of his Applied Mathematics Panel defined the unit “kilogirl,” presumably a term for a thousand hours of computing labor [44]. In this era and beyond, e.g. at NASA, it was a significant and unusual achievement for a woman to get her name on a research report; human computers were even largely excluded from editorial meetings where reports were developed [45]. As such, it was exceptional for Bell Telephone Laboratories Technical Memorandum MM-49-150-29, “Composing Music by a Stochastic Process,” 15 November 1949 to list both John R. Pierce and Mary E. Shannon [46].

Though Pierce and Shannon were unaware at the time, their stochastic music composition algorithms followed in the tradition of W. Mozart, J. Haydn, M. Stadler, and K. P. E. Bach [47], but improved upon it considerably. In the memorandum, the authors introduced stochastic models to describe the generating process of chord progression in four-part harmonies, while carefully controlling the model behavior to resemble what a human composer would do in part writing. The rules for both chord construction (1-gram) and chord progression (2-gram) were largely borrowed from known music theory as well as personal specifications such as keeping common tones between adjacent chords. The set of rules defined the legal actions in the composition process as well as their chances to be selected. Therefore, the resulting chord progression was a realization of the stochastic process specified by the rule set.

Under modern taxonomy, the Pierce-Shannon composition model can be thought of as a probabilistic expert system for symbolic music (not audio waveforms). Both the rules and chances involved in this stochastic model are pre-specified rather than learned from data (a Markov random process rather than reinforcement learning), and thus the algorithm runs mechanically as a sampling automaton once it starts; the algorithm is pre-fixed whereas

its outputs are random realizations. Known weaknesses exist for this composition model, e.g. the composed music lacks a long-range plan due to being Markov and the quality of the results directly depends on the given rules whose design itself is an art. Indeed, as Pierce later described [47], “While the short-range structure of these compositions was very primitive, an effort was made to give them a plausible and reasonably memorable, longer-range structure.... the compositions were primitive rondos.”

In our eyes, the model was a significant achievement in computational creativity, especially for the 1940s when the algorithm was implemented by “throwing three specially made dice and by using a table of random numbers,” [47] and before J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon asserted creativity as a central goal in the formalization of artificial intelligence [48]. Further progress in Markov chain music was made by L. A. Hiller, Jr. and L. M. Isaacson of the University of Illinois at Urbana-Champaign, who formulated the rules of four-part, first-species counterpoint so that an electronic computer could choose notes at random and reject them if rules were violated. This music was curated and published as the *Illiac Suite, for String Quartet* (1957) [49]. Even today, the Pierce-Shannon composition model is used as the core of many modern techniques in music composition and style replication. Perhaps the only change is that modern electronic computers enable extension of the model to higher-order n -grams that allow a long-range composition plan.

In reflecting on their work, John Pierce suggested [47] “the information-theoretic composer... will make up his composition of larger units [recognizable chords, scales, themes, or ornaments] which are already familiar in some degree to listeners through the training they have received in listening to other compositions. ... Perhaps the composer will surprise the listener a bit from time to time, but he won’t try to do this continually.” But where do these concepts, rules, and notions of surprise come from?

Recent research in computer music, including our own [50–53] (which comprises the main body of the music application in this thesis), is concerned with automatically learning a hierarchy of human-interpretable composition rules—the laws of music theory directly from raw sheet music—and best methods to coherently break rules to personalize composition style. As briefly described earlier, our approach to interpretable concept learning uses an iterative alternating optimization of information-theoretic functionals such as Bayesian surprise and mutual information, interpreted as cycling between a generative component (student) and a discriminative component (teacher). The algorithm not only reproduces much of standard music theory in textbooks, but also yields some novel music theory that our music-theorist colleagues find intriguing (Chapter 13). Now, machines may not only execute composition rules automatically, but learn and break rules automatically too. They may even partner with human composers in creative conversations.

Chapter 3: Theoretical Exposition (Part I)

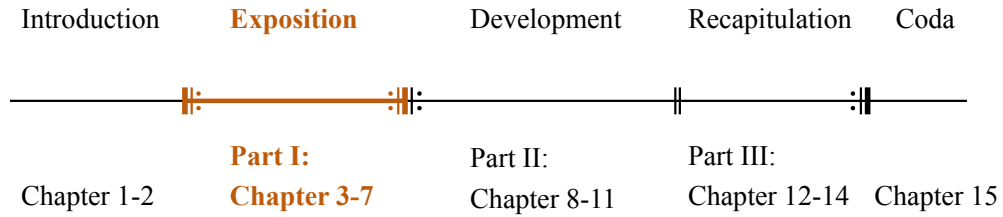


Figure 3.1: Theoretical exposition outline.

This is the beginning chapter of Part I. This part presents the mathematical exposition of ACL, formalizing the key notions of abstraction, concept, probabilistic rule, and further the entire ACL problem (Chapter 4). The whole theory is grounded on a mathematical construct of abstraction, which is then systematically derived towards abstraction hierarchy (Chapter 5) and its generating mechanism (Chapter 6). Note that the goal in mathematizing the notion of abstraction is not for its own sake, but to go from this theoretical base towards computational developments that further bring about feasible, reliable, and human-interpretable algorithmic models.

Chapter 4: Abstraction-Based Rules and Concepts

In this thesis, *abstraction* plays a key role in defining *concepts* and *rules*, the cornerstones that further formalize Automatic Concept Learning (ACL). The former (i.e. the term “concept”) explains the name ACL, and the latter (i.e. the term “rule”) specifies the form of an ACL output. This chapter presents an overall picture of the three key terms—abstraction, concept, rule—and their historical occurrences and usages in various related literatures.

4.1 ABSTRACTION: LITERATURE OVERVIEW

Abstraction describes the process of generalizing high-level concepts from specific data samples by “forgetting the details” [54–56]. This conceptual process is pervasive in human reasoning [3, 9, 10], and its computational counterpart is now also commonly seen in AI and machine learning, with algorithmic models developed to automate abstraction in various concept learning tasks [50, 57–59].

Existing formalizations of abstraction form at least two camps. One uses mathematical logic and reasoning where abstraction is explicitly constructed from abstraction operators and formal languages [15, 57, 60]; the other uses deep learning where abstraction is hinted at by the layered architectures of neural networks [58, 61]. Their key characteristics—commonly known as rule-based (deductive) and data-driven (inductive)—are quite complimentary. The former enjoys model interpretability, but requires explicit handcrafting of complicated logic with massive domain expertise; the latter shifts the burden of model crafting to data, but makes the model less transparent. Perhaps the most famous representatives from the two camps are IBM’s automatic chess player Deep Blue (1997) and DeepMind’s automatic go player AlphaGo (2017), both of which defeated best human players in board games.

This thesis takes a new viewpoint, aiming for a middle ground between the two camps. We formalize abstraction as a mechanism-driven clustering that further admits statistical learning. By clustering, we forget within-cluster variations and discern only between-cluster distinctions [62, 63], revealing the nature of abstraction [29]. In particular, we studied first feature-induced clusterings, and further symmetry-induced clusterings that are more theoretically grounded and require less prior knowledge. While clustering is common in (unsupervised) machine learning [64], our symmetry-induced clustering is in stark contrast with common data clustering settings, as follows.

1. **Data-free.** Our symmetry-induced clustering considers partitioning an input space rather than data samples. It is treated more as conceptual clustering than data cluster-

ing like k -means [65, 66]: clusters are formed in a *mechanism-driven*, not data-driven, fashion; and the mechanisms considered here are *symmetries*. The process is causal, and the results are interpretable. Notably, a single clustering mechanism transfers to multiple domains, and a single clustering result transfers to various data sets.

2. **Feature-free.** Our symmetry-induced clustering involves no feature engineering, so no domain expertise. This particularly means three things. First, no feature design for inputs: we directly deal with mathematical spaces, e.g. vector spaces or manifolds. Second, no feature/assignment function for cluster designation: this differs from algorithms that hand-design abstraction operators [15], arithmetic descriptors (e.g. our initial feature-induced clustering) [50, 51], or decision-tree-like feature thresholding [63]. Third, no meta-feature tuning such as pre-specifying the number of clusters.
3. **Similarity-free.** Our symmetry-induced clustering does not depend on a predefined notion of similarity. This differs from most clustering algorithms where much effort has been expended in defining “closeness” [67, 68]. Instead, pairwise similarity is replaced by an *equivalence relation* induced from symmetry. Note that the definitions of certain symmetries may require additional structure of the input space, e.g. topology or metric, but this is not used as a direct measurement for inverse similarity. Therefore, points that are far apart (in terms of metric distance) in a metric space can be grouped together (in terms of equivalence) under certain symmetries, resulting in a “discontinuous” cluster comprising disconnected regions in the input space. This is not likely to happen for algorithms such as k -means.

It is noteworthy that being feature-free and similarity-free makes a clustering model *universal* [67], becoming more of a *science* than an *art* [69]. Besides the above three distinguishing features, our symmetry-induced clustering exhibits one more distinction regarding hierarchical clustering for multi-level abstractions:

4. **Global hierarchy.** Like many hierarchical clusterings [70, 71], our clustering model outputs a family of multi-level partitions and a hierarchy showing their interrelations. However, here we have a global hierarchy formalized as a *partition (semi)lattice*, which is generated from another hierarchy of symmetries represented by a subgroup lattice. This is in contrast with greedy hierarchical clusterings such as agglomerative/divisive clustering [72, 73] or topological clustering via persistent homology [74]. These greedy algorithms lose many possibilities for clusterings since the hierarchy is constructed by local merges/splits made in a one-directional procedure, e.g. growing a dendrogram

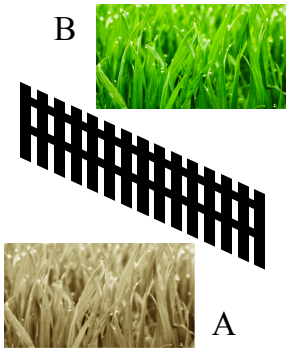
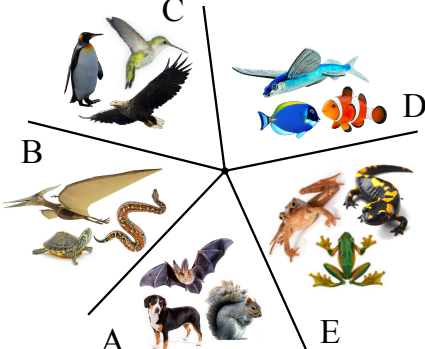
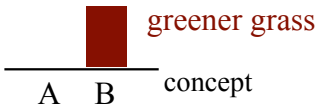

	“The grass is always greener on the other side of the fence.”	“Most birds flying, but rare for fish, amphibians, reptiles, mammals.”
abstraction		
probabilistic rule		

Figure 4.1: Examples of casting real-world rule statements into probabilistic rules.

or a filtration. In particular, greedy hierarchical clustering is oft-criticized since it is hard to recover from bad clusterings in early stages of construction [74]. Lastly, our global hierarchy is represented by a directed acyclic graph rather than tree-like charts (essentially a linear structure) such as dendrograms or barcodes.

4.2 ABSTRACTION, CONCEPT AND PROBABILISTIC RULE

We define an *abstraction* of a set by a partition of the set. Given an abstraction of a set, we define a *concept* by a cell in the partition, and define a (*probabilistic*) *rule* by a pair consisting of the abstraction and a probability distribution over the abstracted concepts. Equivalently, an abstraction of a set can be also viewed from the perspective of an equivalence relation on that set (with the partition being precisely the quotient of the set by the equivalence relation), and accordingly a concept can be also viewed as an equivalence class.

Table 4.1 summarizes the definitions and notations of the three terms. Figure 4.1 gives two examples casting two rule statements (in English sentences) into our abstraction-based probabilistic rules: one shows a deterministic probability distribution over two abstracted concepts {the grass on this side of the fence, the grass on that side of the fence}; the other shows a non-deterministic probability distribution over five abstracted concepts {mammals,

	<i>Definition</i>	<i>Notation</i>
abstraction	partition (equivalence relation)	\mathcal{A}
concept	partition cell (equivalence class)	$C \in \mathcal{A}$
rule	partition & probability distribution	$(\mathcal{A}, p_{\mathcal{A}})$

Table 4.1: Definitions/notations of abstraction, concept, and probabilistic rule.

reptiles, birds, fish, amphibians}

Built on top of abstraction-based concepts and rules, Automatic Concept Learning (ACL) takes as input a data set or a data distribution, and outputs probabilistic rules to summarize and explain the input. Probabilistic rules are further made hierarchal, mechanism-driven, and sequentially disentangled, so as to reveal structured, interpretable, and independent insights of the data.

So far, one might not be immediately clear about the intuitions behind these mathematical definitions, especially the motivations and computational benefits of formalizing an abstraction as a partition. The following chapter will take a steady pace toward our formal derivation of computational abstraction, starting from informal discussions on the nature of abstraction to deeper discussions on abstraction hierarchies, and further towards their algorithmic constructions.

Chapter 5: Computational Abstraction

With the big picture of abstraction and abstraction-based rules and concepts in mind, we begin in this chapter to detail the complete theoretical foundation for computational abstraction, its hierarchies, and its generating mechanisms. It is important to keep in mind that the main purpose of all the machinery developed from here on is threefold (beyond mathematical beauty): to make things precise; to make things interpretable; and to lay down the path towards feasible and efficient algorithms.

5.1 EVERYDAY ABSTRACTION

We first informally discuss abstraction by drawing examples from different disciplines and summarizing their commonalities. Although expressed in everyday terms from specific domains, the intuitions from this section suggest all the key properties of abstraction that the sequel aims to capture formally. In particular, the subsequent sections of this Part I formalize the ideas from this section in a precise and general manner that, importantly, leads to principled algorithmic approaches for automatic concept learning in Part II.

Whether you are aware of it or not, abstraction is everywhere in our daily behaviors. It is in the nature of abstraction that it treats some set of instances that it subsumes as if they were qualitatively identical, although in fact they are not [29]. Examples of people making abstractions can be as simple as observing ourselves through social categories such as race or gender [75]; or as complicated as a systematic taxonomy of a subject domain. In Figure 5.1, we present examples of two systematically derived abstractions: one is from a taxonomy of animals; the other is from a classification of music chords. There are many commonalities in these examples as well as in many other real-life examples of abstraction. We summarize the key properties shared in these abstraction examples, which will be formalized later.

Nature of abstraction: clustering or classification? One shared property among many examples of abstraction is the idea of *clustering* and then forgetting within-cluster variations. For instance, we cluster people into {men, women}, forgetting the difference between John and David, Mary and Rachel; we cluster animals with a backbone into {fish, amphibians, reptiles, birds, mammals}, forgetting the difference between penguins and eagles, bats and dogs; we cluster music triads into {major, minor, augmented, diminished, . . .}, forgetting the difference between C-E-G and F-A-C, C-Eb-G and A-C-E. This idea of clustering is pervasive in various definitions of abstraction, but more often termed as classification (or categorization, taxonomy). Although clustering and classification (likewise clusters and

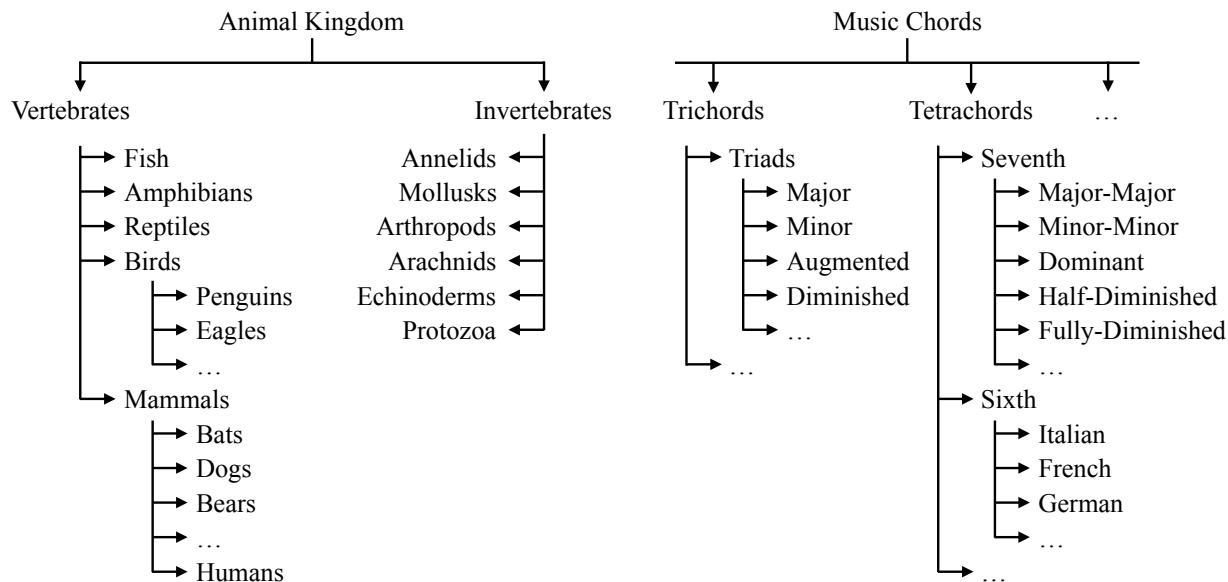


Figure 5.1: Hierarchical abstractions and concepts of Animal Kingdom (left) and music chords (right). Both hierarchies are essentially linear, e.g. kingdom \rightarrow phylum \rightarrow class \rightarrow ... \rightarrow species, which however, is not necessarily the case in general.

classes) are more or less synonyms in everyday life, there is a clear difference between the two in machine learning. The former generally falls under the realm of unsupervised learning, whereas the latter falls under supervised learning. The difference is merely whether or not there is a label for each cluster. Note that labels are important in supervised learning, since a perfect binary classifier with a 100% accuracy is clearly different from a bad one with a 0% accuracy. However, in light of clustering, the two classifiers are identical: the “bad” one, for instance, simply calls all men as women and all women as men, but still accurately captures the concept of gender. Consequently in this thesis, we treat the nature of abstraction as clustering rather than classification, further formalized as a partition or equivalence relation. So, men and women are two equivalence classes of people: all men are equivalent, so are all women. An extended discussion on clustering and classification can be found in Section 5.4, relating to information elements and random variables.

Hierarchy. Another shared property among many examples of abstraction is the presence of a *hierarchy*, where “later” abstractions can be made recursively from “earlier” ones. For instance, we cluster animals into {fish, birds, mammals, annelids, mollusks, ...}, and further cluster these abstracted terms into {vertebrates, invertebrates}; we cluster music chords into {major, minor, dominant, German, ...}, and further cluster these abstracted terms into {triads, seventh chords, sixth chords, ...}, and even further into {trichords, tetrachords, ...}. Hierarchy, being either explicit or implicit, brings the notion of *level of an abstraction*.

For instance, biological taxonomy gives an explicit description of abstraction levels: kindom \rightarrow phylum \rightarrow class \rightarrow order \rightarrow family \rightarrow genus \rightarrow species; whereas the abstraction levels of music chords are relatively implicit but still present. In general, an abstraction hierarchy can be more complicated than simply linear due to various clustering possibilities. In this thesis, a general hierarchy is formalized by a mathematical lattice.

Mechanism. A third shared property among many examples of abstraction is the existence of a *mechanism*—a driving force that causes the resulting abstraction. For instance, the presence or absence of a backbone is the underlying mechanism that results in the abstraction of animals into vertebrates and invertebrates; the intervalic quality is the underlying mechanism that results in the abstraction of music chords. Having a mechanism is important for at least three reasons. First, it makes the abstraction process logical, so that every abstraction is made for a reason. This is a distinguishing feature in human intelligence, which is further key to the development of concepts and knowledge. Second, different mechanisms yield different abstractions, which further yield different attributes of an object. For instance, a bat can be abstracted as a mammal since, among many other reasons, it nurses its pups with milk; a bat can also be abstracted as a flying animal based on its capability of flying. In comparison, under the same two mechanisms, a penguin is abstracted as a bird but flightless. Third, perhaps most importantly, having a mechanism allows generalization, i.e. we can transfer a mechanism from one domain to another. For instance, generalizing the same mechanism under which we abstract people into men and women to other species, we get roosters and hens, bulls and cows, etc. As a result, we emphasize the generating mechanisms for abstractions. In this thesis, we focus on two types of mechanisms: features and symmetries.

Towards laws of nature. Lastly, abstraction is a very important stage towards laws—or less seriously, rules or patterns—of nature [76]. An abstraction itself is not a rule, but an abstraction paired with a property describing that abstraction can be treated as a rule. For instance, the abstraction {fish, amphibians, reptiles, birds, mammals} of animals is not a rule, but a statement like “Most of the birds fly, whereas only a few fish, amphibians, reptiles, or mammals fly” is a rule which indicates what is special about this abstraction. In this thesis, we consider rules made out of abstractions and their statistical properties, which directly gives rise to our definition of probabilistic rules.

5.2 ABSTRACTION AS PARTITION

We formalize an *abstraction process* on a raw data space as solving a clustering problem. In this process, we group elements of the data space (also called data points) into clusters,

so as to deliberately forget or to abstract away within-cluster variations. The outcome is an *abstraction space* whose elements are the clusters denoting concepts. Every abstraction space presents one coarse-grained view of the data space, and different abstraction spaces are possible to present different ways of abstracting the same data space.

Formally, an *abstraction of a set* is a partition of the set, which is a mathematical representation of the outcome of a clustering process. Throughout this thesis, we reserve X to exclusively denote a set which we make abstractions of. The set X can be as intangible as a mathematical space, e.g. \mathbb{R}^n , \mathbb{Z}^n , a manifold in general; or as concrete as a collection of items, e.g. {rat, ox, tiger, rabbit, dragon, snake, horse, sheep, monkey, rooster, dog, pig}.

Keywords and notations: partition of a set (\mathcal{P}), partition cell ($P \in \mathcal{P}$); equivalence relation on a set (\sim), quotient (X/\sim).

Remark 5.1. *An abstraction is a partition, and vice versa. The two terms refer to the same thing, with the only nuance being that one is used less formally, whereas the other is used in the mathematical language. When used as a single noun, these two terms are interchangeable in this thesis.*

Remark 5.2. *A partition is not an equivalence relation. The two terms do not refer to the same thing (one is a set, the other is a binary relation), but convey equivalent ideas since they induce each other bijectively. In this thesis, we use an equivalence relation to explain a partition: elements of a set X are put in the same cell because they are equivalent. Based on this reason, abstracting the set X is about treating equivalent elements as the same, i.e. collapsing equivalent elements in X into a single entity (namely, an equivalence class or a cell) where collapsing is formalized by taking the quotient.*

5.3 ABSTRACTION HIERARCHY AS PARTITION LATTICE

A set X can have multiple partitions, provided that $|X| > 1$. The number of all possible partitions of a set X is called the *Bell number* $B_{|X|}$. Bell numbers grow extremely fast with the size of the set. starting from $B_0 = B_1 = 1$, the first few Bell numbers are:

$$1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, \dots \quad (5.1)$$

We use \mathfrak{P}_X^* to denote the family of all partitions of a set X , so $|\mathfrak{P}_X^*| = B_{|X|}$. We can compare partitions of a set in at least two ways. One simple way is to compare by size: given two partitions \mathcal{P}, \mathcal{Q} of a set, we say that \mathcal{P} is no larger than (resp. no smaller than)

\mathcal{Q} if $|\mathcal{P}| \leq |\mathcal{Q}|$ (resp. $|\mathcal{P}| \geq |\mathcal{Q}|$). Another way of comparison considers the structure of partitions via a *partial order* on \mathfrak{P}_X^* . The partial order further yields a *partition lattice*, a hierarchical representation of a family of partitions.

Keywords and notations: partial order, poset; lattice, join (\vee), meet (\wedge), sublattice, join-semilattice, meet-semilattice, bounded lattice.

Definition 5.1. *Let \mathcal{P} and \mathcal{Q} be two abstractions of a set X . We say that \mathcal{P} is at a higher level than \mathcal{Q} , denoted $\mathcal{P} \preceq \mathcal{Q}$, if as partitions, \mathcal{P} is coarser than \mathcal{Q} . For ease of description, we expand the vocabulary for this definition, so the following are all equivalent:*

1. $\mathcal{P} \preceq \mathcal{Q}$, or equivalently $\mathcal{Q} \succeq \mathcal{P}$ (Figure 5.2).
2. As abstractions, \mathcal{P} is at a higher level than \mathcal{Q} (or \mathcal{P} is an abstraction of \mathcal{Q}).
3. As partitions, \mathcal{P} is coarser than \mathcal{Q} (or \mathcal{P} is a coarsening of \mathcal{Q}).
4. As abstractions, \mathcal{Q} is at a lower level than \mathcal{P} (or \mathcal{Q} is a realization of \mathcal{P}).
5. As partitions, \mathcal{Q} is finer than \mathcal{P} (or \mathcal{Q} is a refinement of \mathcal{P}).
6. Any $x, x' \in X$ in the same cell in \mathcal{Q} are also in the same cell in \mathcal{P} .
7. Any $x, x' \in X$ in different cells in \mathcal{P} are also in different cells in \mathcal{Q} .

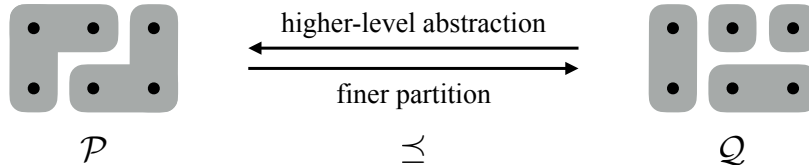


Figure 5.2: The partial order \preceq compares the levels of abstractions.

It is known that the binary relation “coarser than” on the family \mathfrak{P}_X^* of all partitions of a set X is a partial order, so is the binary relation “at a higher level than” on abstractions. Given two partitions \mathcal{P}, \mathcal{Q} of a set, we can have $\mathcal{P} \preceq \mathcal{Q}$, $\mathcal{Q} \preceq \mathcal{P}$, or they are incomparable. Further, $(\mathfrak{P}_X^*, \preceq)$ is a bounded lattice, in which the greatest element is the finest partition $\{\{x\} \mid x \in X\}$ and the least element is the coarsest partition $\{X\}$. For any pair of partitions $\mathcal{P}, \mathcal{Q} \in \mathfrak{P}_X^*$, their join $\mathcal{P} \vee \mathcal{Q}$ is the coarsest common refinement of \mathcal{P} and \mathcal{Q} ; their meet $\mathcal{P} \wedge \mathcal{Q}$ is the finest common coarsening of \mathcal{P} and \mathcal{Q} (Figure 5.3).

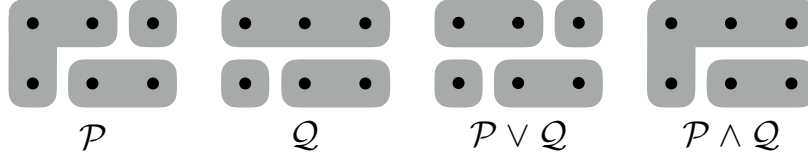


Figure 5.3: Two abstractions \mathcal{P}, \mathcal{Q} and their join $\mathcal{P} \vee \mathcal{Q}$ and meet $\mathcal{P} \wedge \mathcal{Q}$.

Definition 5.2. *An abstraction universe for a set X is a sublattice of \mathfrak{P}_X^* , or a partition (sub)lattice in short. In particular, we call the partition lattice \mathfrak{P}_X^* itself the complete abstraction universe for X . An abstraction join-semiuniverse (resp. meet-semiuniverse) for a set X is a join-semilattice (resp. meet-semilattice) of \mathfrak{P}_X^* . An abstraction family for a set X , an even weaker notion, is simply a subset of \mathfrak{P}_X^* .*

If the complete abstraction universe $(\mathfrak{P}_X^*, \preceq)$ is finite, we can visualize its hierarchy as a directed acyclic graph where vertices denote partitions and edges denote the partial order. The graph is constructed as follows: plot all distinct partitions of X starting at the bottom with the finest partition $\{\{x\} \mid x \in X\}$, ending at the top with the coarsest partition $\{X\}$ and, roughly speaking, with coarser partitions positioned higher than finer ones. Draw edges downwards between partitions using the rule that there will be an edge downward from \mathcal{P} to \mathcal{Q} if $\mathcal{P} \preceq \mathcal{Q}$ and there does not exist a third partition \mathcal{R} such that $\mathcal{P} \preceq \mathcal{R} \preceq \mathcal{Q}$. Thus, if $\mathcal{P} \preceq \mathcal{Q}$, there is a path (possibly many paths) downward from \mathcal{P} to \mathcal{Q} passing through a chain of intermediate partitions (and a path upward from \mathcal{Q} to \mathcal{P} if $\mathcal{Q} \succeq \mathcal{P}$). For any pair of partitions $\mathcal{P}, \mathcal{Q} \in \mathfrak{P}_X^*$, the join $\mathcal{P} \vee \mathcal{Q}$ can be read from the graph as follows: trace paths downwards from \mathcal{P} and \mathcal{Q} respectively until a common partition \mathcal{R} is reached (note that the finest partition $\{\{x\} \mid x \in X\}$ at the bottom is always the end of all downward paths in the graph, so it is guaranteed that \mathcal{R} always exists). To ensure that $\mathcal{R} = \mathcal{P} \vee \mathcal{Q}$, make sure there is no $\mathcal{R}' \preceq \mathcal{R}$ (indicated by an upward path from \mathcal{R} to \mathcal{R}') with upward paths towards both \mathcal{P} and \mathcal{Q} (otherwise replace \mathcal{R} with \mathcal{R}' and repeat the process). Symmetrically, one can read the meet $\mathcal{P} \wedge \mathcal{Q}$ from the graph.

There are limitations to this graph, especially if the set X is infinite. Even for a finite set X of relatively small size, the complete abstraction universe \mathfrak{P}_X^* can be quite complicated to visualize (recall that we have to draw $|\mathfrak{P}_X^*| = B_{|X|}$ vertices where $B_{|X|}$ grows extremely fast with $|X|$, let alone the edges). However, not all arbitrary partitions are of interest to us. In Chapter 6, we will study feature-induced abstractions and symmetry-induced abstractions as well as their corresponding abstraction universes.

5.4 RULE HIERARCHY AS GENERALIZED INFORMATION LATTICE

Recall that a probabilistic rule in an ACL output is defined by a pair consisting of an abstraction and a probability distribution over the abstracted concepts (Section 4.2). Thus, coupling an abstraction hierarchy with a probability measure yields a *hierarchical rule space* or a *rule hierarchy* in short (Figure 5.4), which turns out to be a generalization of Shannon’s information lattice (cf. Section 2.2).

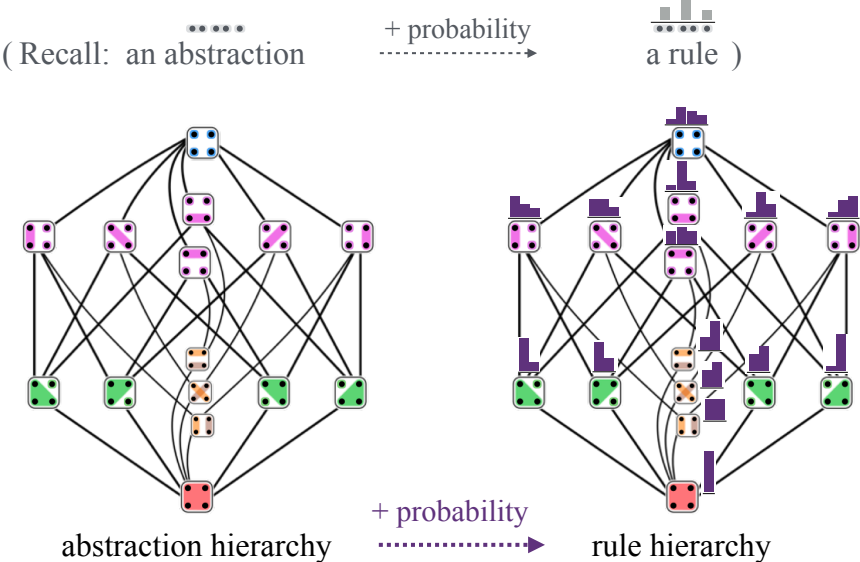


Figure 5.4: From abstraction hierarchy to rule hierarchy: generalizing information lattice.

In particular, we cast Shannon’s information lattice into our abstraction framework, revealing the key difference to be a probability measure:

$$\text{information lattice} = \text{partition lattice} + \text{probability measure.} \tag{5.2}$$

Rewrite our earlier definition of a probabilistic rule:

$$\text{probabilistic rule} = \text{partition} + \text{probability distribution.} \tag{5.3}$$

It is easy to see that (5.2) is a hierarchical generalization of (5.3). This further shows that transferring Shannon’s information lattice originally defined for communication problems to our concept learning problem becomes a hierarchical rule space. Our abstraction-based rule hierarchy reproduces Shannon’s information lattice in more general context, since no information-theoretic functional is needed in the definition. Under this big picture, we discuss in more details about connections between these two parallel worlds.

	<i>Partition lattice</i>	<i>Information lattice</i>
element	partition (\mathcal{P}); clustering (X, \mathcal{P}) ; equiv. class of classifications	information element (x); probability space (X, Σ, P) ; equiv. class of random variables
partial order	$\mathcal{P} \preceq \mathcal{Q}$	$x \leq y \iff H(x y) = 0$
join	$\mathcal{P} \vee \mathcal{Q}$	$x + y$
meet	$\mathcal{P} \wedge \mathcal{Q}$	xy
metric	undefined	$\rho(x, y) = H(x y) + H(y x)$

Table 5.1: Partition lattice and information lattice: the main difference comes from the fact that a partition lattice is not coupled with a measure; whereas an information lattice is coupled with a probability measure, so both the partial order and the metric can be defined in terms of entropies.

Nature of abstraction (clustering or classification?) Generalizing Shannon’s insight on the nature of information essentially reveals the difference between clustering and classification in machine learning. We can similarly define an equivalence relation on the set of all classifications where two classifications are equivalent if they yield the same set of classes and only differ by class labels. For instance, given a set of animals, classifying them into {fish, amphibians, reptiles, birds, mammals} is equivalent to classifying them into {poisson, amphibiens, reptiles, des oiseaux, mammifères}, where the different class labels are only English and French translations of the same set of animal classes. So, clustering to classification is analogous to information element to random variable; and it is clustering rather than classification that captures the nature of abstraction. This once again explains why abstraction is formalized as a clustering problem in this thesis.

Partition lattice and information lattice. By definition, every information element can be uniquely determined by its induced σ -algebra. Also, it is known that every σ -algebra of a countable sample space can be uniquely determined by its generating (via union operation) sample-space-partition. As a result, an information lattice has a one-to-one correspondence to a partition lattice up to different probability measures. We summarize major connections between a partition lattice and an information lattice in Table 5.1. The differences are rooted in the *separation of abstraction (clustering) from statistics*, so roughly speaking, a partition lattice—which is measure-free—can be thought as an information lattice without probability measure, i.e. Equation (5.2). Therefore, abstraction is a more general concept than information, which is not specific to communication problems, and in particular, is not defined via any stochastic processes or information-theoretic functionals such as entropy.

Interpretable rule learning. The separation between partition and probability, or the separation between abstraction and statistics, is important since it opens the opportunity

for *interpretable statistical learning* for probabilistic rules, where *interpretability* is achieved by explicitly constructing a partition lattice from some generating source (e.g. features or subgroups as we will soon see in the Chapter 6), and *learning* is achieved by subsequent statistical inference on this lattice (Chapter 10). This process can be elegantly presented in the following chain aiming for learning:

$$\text{feature/subgroup lattice} \rightarrow \text{partition lattice} \rightarrow \text{information lattice} (\rightarrow \text{learning}). \quad (5.4)$$

At a first glance, regarding subgroup lattices, Chain (5.4) aiming for interpretable learning is merely a reverse process of Chain (2.1) aiming for re-interpretation. However, the subgroup lattices in both chains are in stark contrast: the subgroups considered in Chain (5.4) are based on certain symmetries—the underlying mechanism of abstraction—whereas the subgroups considered in Chain (2.1) are merely (isomorphic) re-statements of the given partitions. In other words, among possibly many subgroups that generate the same partition, we only pick the one that explains to us the types of symmetries under consideration. The preservation of interpretable symmetries through Chain (5.4) makes the subsequent learning transparent. Hence, when abstraction does meet statistics, it will yield interpretable machine learning and knowledge discovery, which is beyond simply a re-interpretation of known results. As such, abstraction generation and statistical learning are the two constituent pillars of our Information Lattice Learning.

Chapter 6: Abstraction Generating Mechanisms

To achieve both the individual (shallow) and the collective (deep) interpretability of a desired ACL output (Section 1.1.3), in this chapter, we discuss abstraction generating mechanisms and their hierarchical interconnections. In particular, we study two types of mechanisms, namely *features* and *symmetries*, yielding *feature-induced abstractions* and *symmetry-induced abstractions*, respectively. Note that this chapter only presents the theory for abstraction generating mechanisms, whereas the corresponding abstraction generating algorithms will be introduced later in Part II.

6.1 FEATURE-INDUCED ABSTRACTION

Conceptually, perhaps the most straightforward generating source of an abstraction is a feature function, where the partition cells are formed by taking the preimages of all possible feature values. For instance, with the feature being color and with the possible feature values being from {red, yellow, blue}, the induced abstraction of a data space simply comprises three concepts (or clusters), namely red data points, yellow data points, and blue data points. It is clear then that the interpretability of a feature-induced abstraction is directly tied to the interpretability of the feature function. When the feature function is conceptually simple or is composed from simple pieces in an interpretable way, we know the abstraction.

Computationally, this is the most efficient way (or at least one of the most efficient ways) of generating an abstraction, since the feature function can be *independently* applied to every data point in the data space to directly give the partition cell that data point belongs to. In particular, the feature value itself is an explicit name or label of that partition cell; thus obviously, the number of all possible feature values is the number of concepts subsumed in the induced abstraction, i.e. the size of the partition.

It is worth noting that, with the above two properties at hand, we are almost in an ideal position here, and this is why we start with features as our first type of abstraction generating mechanisms. Nevertheless, keep in mind that, starting from features is absolutely not necessary for generating abstractions and we are asking for more than needed here. Therefore, there is room for less restrictive mechanisms later (Section 6.2).

6.1.1 From Feature to Abstraction

Mathematically, given a feature function $\phi : X \rightarrow V$ where ϕ is surjective and V denotes the set of all possible feature values, we induce an abstraction \mathcal{P}_ϕ (of X) by taking the set of preimages:

$$\mathcal{P}_\phi := \{\phi^{-1}(\{v\}) \mid v \in \phi(X)\} = X/\sim_\phi \quad (6.1)$$

where corresponding equivalence relation \sim_ϕ is defined as follows: for any $x, x' \in X$, $x \sim_\phi x'$ if and only if $\phi(x) = \phi(x')$, i.e. they are mapped to the same feature value.

Remark 6.1. *A feature function on a set X provides more information than a partition of the set X ; in particular, it is common to have two distinct feature functions $\phi \neq \phi'$ but $\mathcal{P}_\phi = \mathcal{P}_{\phi'}$. As a result, constructing an abstraction from a feature function can be thought of as obtaining a clustering from a classification where the feature function is treated as the class function. Because of the redundancy and the non-necessity of the feature values (i.e. class labels), we will introduce a more general framework to construct abstractions, namely symmetry-induced abstraction in Section 6.2.*

6.1.2 From a Feature Pool to an Abstraction Family

Given a pool of feature functions on a common domain X , say Φ , we can generate a family of abstractions of X by

$$\mathfrak{P}_X^\Phi := \{\mathcal{P}_\phi \mid \phi \in \Phi\} \subseteq \mathfrak{P}_X^*. \quad (6.2)$$

Instead of considering arbitrary feature functions, we focus on the feature pool that is spanned by a finite set of basis features that are individually “simple” (e.g. basic arithmetic operators like `sort` and `mod`) and thus easy for people to interpret. Let B denote a finite set of such basis features, we systematically construct the depth- k feature pool spanned by B as follows:

$$\Phi^{[k]} := \{b_{k'} \circ \dots \circ b_1 \mid b_{k'}, \dots, b_1 \in B, 0 \leq k' \leq k\}. \quad (6.3)$$

We follow the convention that $b_{k'} \circ \dots \circ b_1 = \text{id}$ (the identity function) for $k' = 0$. So, $\text{id} \in \Phi^{[k]}$ for all possible B and for all $k \geq 0$, and it is worth noting that $\mathcal{P}_{\text{id}} = \{\{x\} \mid x \in X\}$, i.e. the finest partition of the set X .

In light of self-exploration (the first desired model property in Section 1.1.4), we want to

pre-select a set of basis features that encode as little domain knowledge as possible, but the spanned $\Phi^{[k]}$ is expressive enough to capture much of what we know. The key idea here is the ability to break a rich pool of domain-specific features into a set of domain-agnostic basis features as building blocks. This idea is analogous to creating various meaningful figures from seven basic (meaningless) shapes in a tangram or composing various meaningful English articles out of 26 (meaningless) English letters and a finite set of (meaningless) punctuation marks. The design of such a set of basis features will be exemplified in Part III.

6.2 SYMMETRY-INDUCED ABSTRACTION

As already mentioned in Section 6.1, generating abstractions from feature functions is asking for more than needed, which is essentially obtaining clustering from classification. Further, regarding a hierarchical family of feature-induced abstractions, the design of basis features require a bit of domain intuition, and the generated family is at high risk of containing many duplicates that are not easy to detect before their actual generations. These motivate us to study a more general type of mechanisms, namely symmetries, and accordingly, a more general framework for abstraction generation, namely symmetry-induced abstractions. As the name suggests, a symmetry-induced abstraction is a partition induced by some symmetry, where the partition cells are invariant with respect to the symmetry.

6.2.1 From Symmetry to Abstraction

To capture various symmetries, we consider groups and group actions.

Keywords and notations: group ($(G, *)$ or G), subgroup (\leq), trivial subgroup ($\{e\}$), subgroup generated by a set ($\langle S \rangle$), cyclic subgroup ($\langle s \rangle$); group action, G -action on a set X ($\cdot : G \times X \rightarrow X$), orbit of $x \in X$ (Gx), set of all orbits (X/G).

Consider the symmetric group (S_X, \circ) defined over a set X , whose group elements are all the bijections from X to X and whose group operation is (function) composition. The identity element of S_X is the identity function, denoted id . A bijection from X to X is also called a *transformation* of X . Therefore, the symmetric group S_X comprises all transformations of X , and is also called the transformation group of X , denoted $F(X)$. We use these two terms and notations interchangeably in this thesis, with a preference for $F(X)$ in general, while reserving S_X mostly for a finite X .

Given a set X and a subgroup $H \leq F(X)$, we define an H -action on X by $h \cdot x := h(x)$ for any $h \in H, x \in X$; the *orbit* of $x \in X$ under H is the set $Hx := \{h(x) \mid h \in H\}$.

Orbits in X under H define an equivalence relation: $x \sim x'$ if and only if x, x' are in the same orbit, and each orbit is an equivalence class. Thus, the quotient $X/H = X/\sim$ is a partition of X . It is known that every cell (or orbit) in the abstraction (or quotient) X/H is a minimal non-empty invariant subset of X under transformations in H . We say this abstraction respects the so-called *H-symmetry* or *H-invariance*.

We succinctly record the above process of constructing an abstraction X/H (of X) from a given subgroup $H \leq \mathbf{F}(X)$ in the following *abstraction generating chain*:

$$\text{a subgroup of } \mathbf{F}(X) \xrightarrow{\text{group action}} \text{orbits} \xrightarrow{\text{equiv. rel.}} \text{a partition} \xrightarrow{\text{is}} \text{an abstraction of } X, \quad (6.4)$$

which can be further encapsulated by the *abstraction generating function* defined as follows.

Definition 6.1. *The abstraction generating function is the mapping $\pi : \mathcal{H}_{\mathbf{F}(X)}^* \rightarrow \mathfrak{P}_X^*$ where $\mathcal{H}_{\mathbf{F}(X)}^*$ is the collection of all subgroups of $\mathbf{F}(X)$, \mathfrak{P}_X^* is the family of all partitions of X , and for any $H \in \mathcal{H}_{\mathbf{F}(X)}^*$, $\pi(H) := X/H := \{Hx \mid x \in X\}$, where $Hx := \{h(x) \mid h \in H\}$.*

Theorem 6.1. *The abstraction generating function $\pi : \mathcal{H}_{\mathbf{F}(X)}^* \rightarrow \mathfrak{P}_X^*$ is not necessarily injective.*

Proof. Let $X = \{1, 2, 3, 4\}$ and $h = (1234), g = (1324) \in S_4 = \mathbf{F}(X)$ be two transformations (also known as permutations, in the cycle notation) of X ; consider the cyclic groups:

$$H = \langle h \rangle := \{h^n \mid n \in \mathbb{Z}\} = \{\text{id}, h, h^2, h^3\} = \{\text{id}, (1234), (13)(24), (1432)\}; \quad (6.5)$$

$$G = \langle g \rangle := \{g^n \mid n \in \mathbb{Z}\} = \{\text{id}, g, g^2, g^3\} = \{\text{id}, (1324), (12)(34), (1423)\}. \quad (6.6)$$

It is clear that $H \neq G$ but $\pi(H) = \pi(G) = \{\{1, 2, 3, 4\}\}$, the coarsest partition of X .

Theorem 6.2. *The abstraction generating function $\pi : \mathcal{H}_{\mathbf{F}(X)}^* \rightarrow \mathfrak{P}_X^*$ is surjective.*

Proof. For any $a, b \in X$, let $f_{a,b} : X \rightarrow X$ be the bijective function of the form

$$f_{a,b}(x) = \begin{cases} a & x = b, \\ b & x = a, \\ x & \text{otherwise.} \end{cases} \quad (6.7)$$

Pick any partition $\mathcal{P} \in \mathfrak{P}_X^*$. For any cell $P \in \mathcal{P}$, define

$$S_P := \{f_{a,b} \mid a, b \in P, a \neq b\}, \quad \text{and let } H := \left\langle \bigcup_{P \in \mathcal{P}} S_P \right\rangle. \quad (6.8)$$

We claim $\pi(H) = \mathcal{P}$. To see this, for any distinct $x, x' \in X$ that are in the same cell in \mathcal{P} , $f_{x,x'} \in S_P$ for some $P \in \mathcal{P}$, so $f_{x,x'} \in H$. This implies that x and x' are in the same orbit in $\pi(H)$, since $x' = f_{x,x'}(x)$. Therefore, $\pi(H) \preceq \mathcal{P}$. Conversely, for any distinct $x, x' \in X$ that are in the same orbit in $\pi(H)$, there exists an $h \in H$ such that $x' = h(x)$. By definition, $h = h_k \circ \dots \circ h_1$ for some finite integer $k > 0$ where $h_k, \dots, h_1 \in \cup_{P \in \mathcal{P}} S_P$. Suppose $P' \in \mathcal{P}$ is the cell that x is in, i.e. $x \in P'$, then $h_1(x) \in P'$, since $h_1(x) \in P'$ if $h_1 \in S_{P'}$ and $h_1(x) = x$ otherwise. Likewise, we have $h_2 \circ h_1(x), h_3 \circ h_2 \circ h_1(x), \dots, h_k \circ \dots \circ h_1(x) \in P'$. This implies that $x' = h(x) = h_k \circ \dots \circ h_1(x) \in P'$, i.e. x and x' are in the same cell in \mathcal{P} . Therefore, $\mathcal{P} \preceq \pi(H)$. Combining both directions yields $\pi(H) = \mathcal{P}$, so π is surjective.

6.2.2 Duality: from Subgroup Lattice to Abstraction (Semi)Universe

Given a subgroup of $F(X)$, we can generate an abstraction of X via the abstraction generating function π . So, given a collection of subgroups of $F(X)$, we can generate a family of abstractions of X . Further, given a collection of subgroups of $F(X)$ with a hierarchy, we can generate a family of abstractions of X with an induced hierarchy. This leads us to a *subgroup lattice* generating a partition (semi)lattice, where the latter is dual to the former via the abstraction generating function π .

Keywords and notations: the (complete) subgroup lattice for a group (\mathcal{H}_G^*, \leq) , join ($A \vee B = \langle A \cup B \rangle$), meet ($A \wedge B = A \cap B$).

We consider the subgroup lattice for $F(X)$, denoted $(\mathcal{H}_{F(X)}^*, \leq)$. Similar to the complete abstraction universe $(\mathfrak{P}_X^*, \preceq)$, we can draw a directed acyclic graph to visualize $(\mathcal{H}_{F(X)}^*, \leq)$ if it is finite, where vertices denote subgroups and edges denote the partial order. The graph is similarly constructed by plotting all distinct subgroups of $F(X)$ starting at the bottom with $\{\text{id}\}$, ending at the top with $F(X)$ and, roughly speaking, with larger subgroups positioned higher than smaller ones. Draw an upward edge from A to B if $A \leq B$ and there are no subgroups properly between A and B . For any pair of subgroups $A, B \in \mathcal{H}_{F(X)}^*$, the join $A \vee B$ can be read from the graph by tracing paths upwards from A and B respectively until a common subgroup containing both is reached, and making sure there are no smaller such subgroups; the meet $A \wedge B$ can be read from the graph in a symmetric manner. For any subgroup $C \in \mathcal{H}_{F(X)}^*$, the subgroup sublattice (\mathcal{H}_C^*, \leq) for C is part of the subgroup lattice $(\mathcal{H}_{F(X)}^*, \leq)$ for $F(X)$, which can be read from the graph for $(\mathcal{H}_{F(X)}^*, \leq)$ by extracting the part below C and above $\{\text{id}\}$.

Theorem 6.3 (Duality). *Let $(\mathcal{H}_{F(X)}^*, \leq)$ be the subgroup lattice for $F(X)$, and π be the abstraction generating function. Then $(\pi(\mathcal{H}_{F(X)}^*), \preceq)$ is an abstraction meet-semiuniverse*

for X . More specifically, for any $A, B \in \mathcal{H}_{\mathbb{F}(X)}^*$, the following hold:

1. *partial-order reversal*: if $A \leq B$, then $\pi(A) \succeq \pi(B)$;
2. *strong duality*: $\pi(A \vee B) = \pi(A) \wedge \pi(B)$ (Figure 6.1a);
3. *weak duality*: $\pi(A \wedge B) \succeq \pi(A) \vee \pi(B)$ (Figure 6.1b).

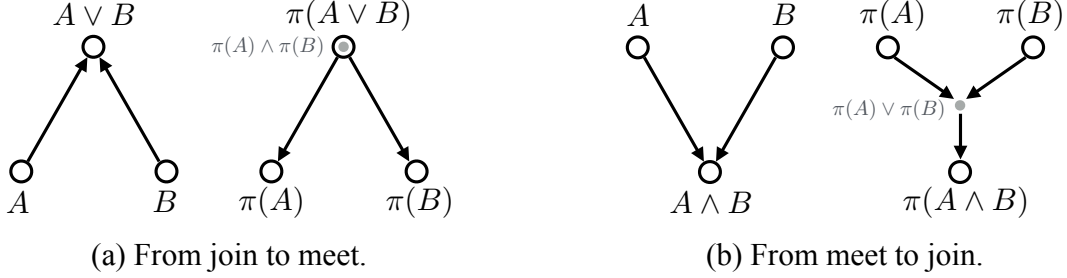


Figure 6.1: Duality of join and meet between the subgroup lattice (left in each subfigure) and the partition lattice (right in each subfigure). In (a), the gray vertex denoting $\pi(A) \wedge \pi(B)$, i.e. the actual meet in the partition lattice, is equal to $\pi(A \vee B)$; in (b), the gray vertex denoting $\pi(A) \vee \pi(B)$, i.e. the actual join in the partition lattice, can be any vertex below $\pi(A), \pi(B)$ and above $\pi(A \wedge B)$ or even equal to these three end points.

Proof. (Partial-order reversal) Pick any $A, B \in \mathcal{H}_G^*$ and $A \leq B$. For any $x, x' \in X$ that are in the same cell in partition $\pi(A) = X/A = \{Ax \mid x \in X\}$, $x' \in Ax = \{a(x) \mid a \in A\}$. Since $A \leq B$, then $Ax \subseteq Bx$, which further implies that $x' \in Bx$. So, x and x' are in the same cell in partition $\pi(B)$. Therefore, $\pi(A) \succeq \pi(B)$.

(Strong duality) Pick any $A, B \in \mathcal{H}_G^*$. By the definition of join, $A, B \leq A \vee B$, so from what we have shown at the beginning, $\pi(A), \pi(B) \succeq \pi(A \vee B)$, i.e. $\pi(A \vee B)$ is a common coarsening of $\pi(A)$ and $\pi(B)$. Since $\pi(A) \wedge \pi(B)$ is the finest common coarsening of $\pi(A)$ and $\pi(B)$, then $\pi(A \vee B) \preceq \pi(A) \wedge \pi(B)$. Conversely, for any $x, x' \in X$ that are in the same cell in partition $\pi(A \vee B) = \pi(\langle A \cup B \rangle) = X/\langle A \cup B \rangle = \{\langle A \cup B \rangle x \mid x \in X\}$, x and x' must be in the same orbit under $\langle A \cup B \rangle$ -action on X , i.e. $x' \in \langle A \cup B \rangle x$ which means $x' = f_k \circ \dots \circ f_1(x)$ for some finite integer k where $f_1, \dots, f_k \in A \cup B$ (note: the fact that A, B are both subgroups ensures that $A \cup B$ is closed under inverses). This implies that x and $f_1(x)$ are either in the same cell in partition $\pi(A)$ or in the same cell in partition $\pi(B)$ depending on whether $f_1 \in A$ or $f_1 \in B$, but in either event, x and $f_1(x)$ must be in the same cell in any common coarsening of $\pi(A)$ and $\pi(B)$. Note that $\pi(A) \wedge \pi(B)$ is a common coarsening of $\pi(A)$ and $\pi(B)$ (regardless of the fact that it is the finest), so x and $f_1(x)$ are in the same cell in partition $\pi(A) \wedge \pi(B)$. Likewise, $f_1(x)$ and $f_2 \circ f_1(x), f_3 \circ f_2 \circ f_1(x)$ and

$f_2 \circ f_1(x), \dots, f_{k-1} \circ \dots \circ f_1(x)$ and x' are all in the same cell in partition $\pi(A) \wedge \pi(B)$. Therefore, x and x' are in the same cell in partition $\pi(A) \wedge \pi(B)$. So, $\pi(A \vee B) \succeq \pi(A) \wedge \pi(B)$. Combining both directions yields $\pi(A \vee B) = \pi(A) \wedge \pi(B)$.

(Weak duality) Pick any $A, B \in \mathcal{H}_G^*$. By the definition of meet, $A, B \geq A \wedge B$, so from what we have shown at the beginning, $\pi(A), \pi(B) \preceq \pi(A \wedge B)$, i.e. $\pi(A \wedge B)$ is a common refinement of $\pi(A)$ and $\pi(B)$. Since $\pi(A) \vee \pi(B)$ is the coarsest common refinement of $\pi(A)$ and $\pi(B)$, then $\pi(A \wedge B) \succeq \pi(A) \vee \pi(B)$. We cannot obtain equality in general. For example, let $X = \mathbb{Z}$ and $A = \{r : \mathbb{Z} \rightarrow \mathbb{Z} \mid r(x) = kx, k \in \{-1, 1\}\}$, $B = \{t : \mathbb{Z} \rightarrow \mathbb{Z} \mid t(x) = x + k, k \in \mathbb{Z}\}$. It is clear that $A, B \leq F(X)$ and $A \wedge B = A \cap B = \{\text{id}\}$, so $\pi(A \wedge B) = X/\{\text{id}\} = \{\{x\} \mid x \in \mathbb{Z}\}$, i.e. the finest partition of \mathbb{Z} . However, $\pi(A) = \{\{x, -x\} \mid x \in \mathbb{Z}\}$ and $\pi(B) = \{\mathbb{Z}\}$, i.e. the coarsest partition of \mathbb{Z} , so $\pi(A) \vee \pi(B) = \pi(A) = \{\{x, -x\} \mid x \in \mathbb{Z}\}$. In this example, we see that $\pi(A \wedge B) \succeq \pi(A) \vee \pi(B)$ but $\pi(A \wedge B) \neq \pi(A) \vee \pi(B)$.

Remark 6.2 (Practical implication). *The strong duality in Theorem 6.3 suggests a quick way of computing abstractions. If one has already computed abstractions $\pi(A)$ and $\pi(B)$, then instead of computing $\pi(A \vee B)$ from $A \vee B$, one can compute the meet $\pi(A) \wedge \pi(B)$, which is generally a less expensive operation than computing $A \vee B$ and identifying all orbits in $\pi(A \vee B)$.*

Theorem 6.3 further allows us to build an abstraction semiuniverse with a partial hierarchy directly inherited from the hierarchy of the subgroup lattice. Nevertheless, there are cases where $\pi(A) \preceq \pi(B)$ with incomparable A and B since the abstraction generating function π is not injective (Theorem 6.1). If desired, one needs additional steps to complete the hierarchy or even to complete the abstraction semiuniverse into an abstraction universe.

6.2.3 More on Duality: from Conjugation to Group Action

Partitions of a set X generated from two conjugate subgroups of $F(X)$ can be related by a group action. We present this relation as another duality between subgroups and abstractions, which can also simplify the computation of abstractions.

Keywords and notations: conjugate, conjugacy class.

Theorem 6.4. *Let G be a group, X be a set, and $\cdot : G \times X \rightarrow X$ be a G -action on X . Then*

1. *for any $g \in G, Y \in 2^X$, $g \cdot Y := \{g \cdot y \mid y \in Y\} \in 2^X$, and the corresponding function $\cdot : G \times 2^X \rightarrow 2^X$ defined by $g \cdot Y$ is a G -action on 2^X ;*

2. for any $g \in G, \mathcal{P} \in \mathfrak{P}_X^*, g \cdot \mathcal{P} := \{g \cdot P \mid P \in \mathcal{P}\} \in \mathfrak{P}_X^*$, and the corresponding function $\cdot : G \times \mathfrak{P}_X^* \rightarrow \mathfrak{P}_X^*$ defined by $g \cdot \mathcal{P}$ is a G -action on \mathfrak{P}_X^* .

Proof. Pick any $g \in G$ and $Y \in 2^X$. For any $x \in g \cdot Y$, we have $x = g \cdot y$ for some $y \in Y$. Since $Y \in 2^X$, i.e. $Y \subseteq X$, then $y \in X$. This implies that $x = g \cdot y \in X$. Therefore, $g \cdot Y \subseteq X$, i.e. $g \cdot Y \in 2^X$. To see the corresponding function $\cdot : G \times 2^X \rightarrow 2^X$ is a G -action on 2^X , we first check that the identity element $e \in G$ satisfies $e \cdot Y = \{e \cdot y \mid y \in Y\} = \{y \mid y \in Y\} = Y$; then check that for any $g, h \in G, g \cdot (h \cdot Y) = \{g \cdot z \mid z \in \{h \cdot y \mid y \in Y\}\} = \{g \cdot (h \cdot y) \mid y \in Y\} = \{(gh) \cdot y \mid y \in Y\} = (gh) \cdot Y$.

Pick any $g \in G$ and $\mathcal{P} \in \mathfrak{P}_X^*$. For any distinct elements $Q, Q' \in g \cdot \mathcal{P}$, we have $Q = g \cdot P$ and $Q' = g \cdot P'$ for some distinct $P, P' \in \mathcal{P}$, respectively. Since P, P' are two distinct cells in partition \mathcal{P} , $P \cap P' = \emptyset$. We claim that $Q \cap Q' = \emptyset$. Assume otherwise, then there exists a $q \in Q \cap Q'$ and $q = g \cdot p = g \cdot p'$ for some $p \in P, p' \in P'$. This implies that $p = (g^{-1}g) \cdot p = g^{-1} \cdot (g \cdot p) = g^{-1} \cdot (g \cdot p') = (g^{-1}g) \cdot p' = p' \in P \cap P'$, which contradicts the fact that $P \cap P' = \emptyset$. For any $x \in X, g^{-1} \cdot x \in X$, then there exists a cell $P \in \mathcal{P}$ such that $g^{-1} \cdot x \in P$. This implies that $x = (gg^{-1})x = g \cdot (g^{-1} \cdot x) \in g \cdot P$ which is an element in $g \cdot \mathcal{P}$. Therefore, the union of all elements in $g \cdot \mathcal{P}$ covers X , or more precisely, equals X , since every element in $g \cdot \mathcal{P}$ is a subset of X . Hence, $g \cdot \mathcal{P}$ is indeed a partition of X , i.e. $g \cdot \mathcal{P} \in \mathfrak{P}_X^*$. To see the corresponding function $\cdot : G \times \mathfrak{P}_X^* \rightarrow \mathfrak{P}_X^*$ is a G -action on \mathfrak{P}_X^* , we first check that the identity element $e \in G$ satisfies $e \cdot \mathcal{P} = \{e \cdot P \mid P \in \mathcal{P}\} = \{P \mid P \in \mathcal{P}\} = \mathcal{P}$; then check that for any $g, h \in G, g \cdot (h \cdot \mathcal{P}) = \{g \cdot Q \mid Q \in \{h \cdot P \mid P \in \mathcal{P}\}\} = \{g \cdot (h \cdot P) \mid P \in \mathcal{P}\} = \{(gh) \cdot P \mid P \in \mathcal{P}\} = (gh) \cdot \mathcal{P}$.

Theorem 6.5 (Duality). *Let X be a set, $\mathbf{F}(X)$ be the transformation group of X , and π be the abstraction generating function. Then for any $H \leq \mathbf{F}(X)$ and $g \in \mathbf{F}(X)$,*

$$\pi(g \circ H \circ g^{-1}) = g \cdot \pi(H). \quad (6.9)$$

where \cdot refers to the group action defined in Statement 2 in Theorem 6.4.

Proof. For any $Y \in \pi(g \circ H \circ g^{-1})$, Y is an orbit in X under $g \circ H \circ g^{-1}$, then $Y = (g \circ H \circ g^{-1})x = \{(g \circ h \circ g^{-1}) \cdot x \mid h \in H\} = \{g \circ h \circ g^{-1}(x) \mid h \in H\} = \{(g \circ h)(g^{-1}(x)) \mid h \in H\} = \{(gh) \cdot g^{-1}(x) \mid h \in H\} = \{g \cdot (h \cdot g^{-1}(x)) \mid h \in H\} = \{g \cdot y \mid y \in Hg^{-1}(x)\} = g \cdot Hg^{-1}(x)$ for some $x \in X$. Note that in the above derivation, $g^{-1}(x) \in X$ since $g \in \mathbf{F}(X)$. So, $Hg^{-1}(x)$ is the orbit of $g^{-1}(x)$ under H , i.e. $Hg^{-1}(x) \in \pi(H)$. This implies that $Y \in g \cdot \pi(H)$. Therefore, $\pi(g \circ H \circ g^{-1}) \subseteq g \cdot \pi(H)$.

Conversely, for any $Y \in g \cdot \pi(H)$, $Y = g \cdot P$ for some $P \in \pi(H)$. Note that P is an orbit in X under H , i.e. $P = Hx = \{h \cdot x \mid h \in H\}$ for some $x \in X$, then $Y = g \cdot P = \{g \cdot y \mid y \in$

$P\} = \{g \cdot (h \cdot x) \mid h \in H\} = \{(gh) \cdot x \mid h \in H\} = \{g \circ h(x) \mid h \in H\} = \{g \circ h \circ g^{-1} \circ g(x) \mid h \in H\} = \{(g \circ h \circ g^{-1})(g(x)) \mid h \in H\} = \{(g \circ h \circ g^{-1}) \cdot g(x) \mid h \in H\} = (g \circ H \circ g^{-1})g(x)$
for some $x \in X$. Note that in the above derivation, $g(x) \in X$ since $g \in F(X)$. Therefore, $(g \circ H \circ g^{-1})g(x)$ is the orbit of $g(x)$ under $g \circ H \circ g^{-1}$, i.e. $(g \circ H \circ g^{-1})g(x) \in \pi(g \circ H \circ g^{-1})$. This implies that $Y \in \pi(g \circ H \circ g^{-1})$. So, $g \cdot \pi(H) \subseteq \pi(g \circ H \circ g^{-1})$.

Remark 6.3 (Practical implication). *Theorem 6.5 relates conjugation in the subgroup lattice $\mathcal{H}_{F(X)}^*$ to group action on the partition lattice \mathfrak{P}_X^* . In other words, the group action on the partition lattice is dual to the conjugation in the subgroup lattice. This duality suggests a quick way of computing abstractions. If one has already computed abstraction $\pi(H)$, then instead of computing $\pi(g \circ H \circ g^{-1})$ from $g \circ H \circ g^{-1}$, one can compute $g \cdot \pi(H)$, which is generally a less expensive operation than computing $g \circ H \circ g^{-1}$ and identifying all orbits in $\pi(g \circ H \circ g^{-1})$.*

6.2.4 Partial Subgroup Lattice

Theoretically, through the abstraction generating function π and necessary hierarchy completions, we can construct the complete abstraction universe \mathfrak{P}_X^* from the complete subgroup lattice $\mathcal{H}_{F(X)}^*$. This is because the subgroup lattice is a larger space that “embeds” the partition lattice (more precisely, Theorem 6.1 and 6.2). However, as we mentioned earlier, it is not practical to even store \mathfrak{P}_X^* for small X , and not all arbitrary partitions of X are equally useful. Instead of considering all subgroups of $F(X)$, we draw our attention to certain parts of the complete subgroup lattice $\mathcal{H}_{F(X)}^*$. In the first half of Part II, we will introduce two general algorithmic principles in extracting partial subgroup lattices and provide the corresponding algorithms; yet as a quick preview here, we take a glimpse through the main ideas of the two principles, namely the top-down approach and the bottom-up approach.

The Top-Down Approach. We consider the subgroup sublattice (\mathcal{H}_G^*, \leq) for some subgroup $G \leq F(X)$. If X is finite, this is the part below G and above $\{\text{id}\}$ in the directed acyclic graph for the complete subgroup lattice $(\mathcal{H}_{F(X)}^*, \leq)$. As the name suggests, the top-down approach first specifies a “top” in $\mathcal{H}_{F(X)}^*$ (i.e. a subgroup $G \leq F(X)$), and then extract everything below the “top” (i.e. the subgroup lattice \mathcal{H}_G^*). The computer algebra system GAP [77] provides efficient algorithmic methods to construct the subgroup lattice for a given group, and even maintains several data libraries for special groups and their subgroup lattices. In general, enumerating all subgroups of a group can be computationally intense, and therefore, is applied primarily to small groups. When computationally prohibited, a general trick is to enumerate subgroups up to conjugacy (which is also supported by the

GAP system). Computing abstractions within the conjugacy class of any subgroup is then easy by the duality in Theorem 6.5, once the abstraction generated by a representative is computed. More details on picking a special subgroup (as the “top”) of $F(X)$ are discussed in Section 9.2, Part II.

The Bottom-Up Approach. We first pick some finite subset $S \subseteq F(X)$, and then generate a partial subgroup lattice for $\langle S \rangle$ by computing $\langle S' \rangle$ for every $S' \subseteq S$, starting from smaller subgroups. As the name suggests, the bottom-up approach first constructs the trivial subgroup $\langle \emptyset \rangle = \{\text{id}\}$, i.e. the bottom vertex in the direct acyclic graph for $\mathcal{H}_{F(X)}^*$ if X is finite, and then cyclic subgroups $\langle s \rangle$ for every $s \in S$. We continue to construct larger subgroups from smaller ones by taking the join, which corresponds to gradually moving upwards in the graph for $\mathcal{H}_{F(X)}^*$ when X is finite. In general, this approach will produce at most $2^{|S|}$ subgroups for a given subset $S \subseteq F(X)$, and will not produce the complete subgroup sublattice $\mathcal{H}_{\langle S \rangle}^*$ unless $S = \langle S \rangle$. Computing abstractions using this bottom-up approach is easy by the strong duality in Theorem 6.3, once the abstractions generated by all cyclic subgroups are computed. More details on this abstraction generating process and picking a generating set (as the “bottom”) are discussed in Section 9.3, Part II.

Chapter 7: Codetta: Summary and Discussions

Part I formalizes the definitions of many terms in Automatic Concept Learning (ACL) as well as several important notions of a desired model and model output mentioned in Section 1.1. The introduced formalism coincides well with the intuition behind it. However, what is more important is that among many other possible formalization choices, the proposed concept learning formalism establishes the playground upon which a fully automatic concept learning is possible. We will elaborate this fully automated learning cycle as the second half in Part II, with the first half being algorithmic generation of computational abstractions. The two halves are the two constituent pillars of Information Lattice Learning (ILL), our new learning model for ACL.

The current theoretical foundation for ACL focuses on conceptualizing a static data space, whose formalism is woven around the core idea of hierarchical abstractions of individual data points. One big next move is to extend the current formalism towards the goal of conceptualizing a dynamic process, considering hierarchical abstractions of data sequences. This is similar to moving from Convolutional Neural Networks (CNNs) for static data to Recurrent Neural Networks (RNNs) for time-series data in deep learning. Enabling *abstraction through time* will bring about a new modeling perspective in which a sequential process unfolds hierarchically from global plans to local plans rather than linearly (or sometimes bi-linearly) from the past to the present (or sometimes also from the future to the present). Take music as an example: under this new viewpoint, the composition of a piece is modeled more naturally as a hierarchical decision process from its global form, to sections, to phrases, and eventually to local melodic and harmonic realizations to flesh out those more global abstractions. This is in contrast with our current approach used later in MUS-ROVER (Chapter 13), our automatic concept learner in music, which simply employs n -gram models to describe *transitions of the abstractions*, but lacks *abstractions of the transitions*.

Chapter 8: Algorithmic Development (Part II)

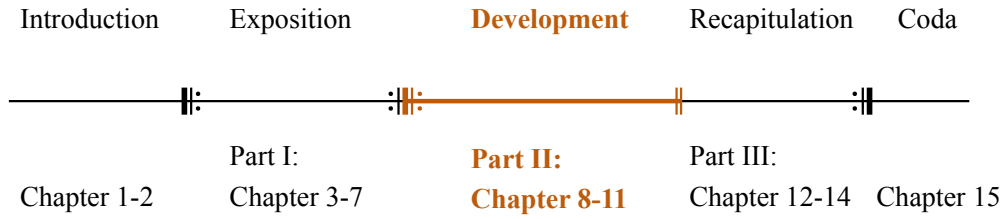


Figure 8.1: Algorithmic development outline.

This is the beginning chapter of Part II. This part presents the algorithmic development of ACL, proposing the new learning model ILL. ILL includes two phases, with the first phase—abstraction generation (Chapter 9)—being the deductive preparation phase and the second phase—probabilistic rule learning (Chapter 10)—being the subsequent inductive learning phase. The two phases are the constituent pillars of ILL, which couples abstraction with statistics to fill in the gap between a rule-based AI and a machine learning AI. The resulting ILL model achieves a deeper-level interpretability, wherein not only the learned results (i.e. rules and concepts) are interpretable, but also the entire leaning process (i.e. the model itself) is comprehensible to people.

Chapter 9: Information Lattice Learning Phase I: Abstraction Generation

We present algorithmic guidelines for constructing abstractions, the first phase of our two-phase Information Lattice Learning (ILL). The output of any abstraction generation algorithm in this chapter is an abstraction (semi)universe including both the abstractions and the hierarchy. The produced abstraction hierarchy will be further used in the subsequent probabilistic rule learning (Chapter 10), the second phase of ILL that completes the entire automatic concept learning.

In this chapter, Section 9.1 discusses feature generation algorithms used for feature-induced abstractions. Sections 9.2 and 9.3 discuss symmetry generation algorithms used for symmetry-induced abstractions, with the former adopting a top-down approach and the latter adopting a bottom-up approach.

9.1 FEATURE GENERATION

We consider the input data space to be the n -dimensional Euclidean space: $X = \mathbb{R}^n$, so every data point $x \in X$ is an n -dimensional vector, a standard setting in machine learning. To systematically generate a large and expressive pool of candidate features, we start with two types of basis features on X , namely (selection) windows and (basis) descriptors, and then use them to span a feature pool via function composition.

A *selection window* $w_I : \mathbb{R}^n \rightarrow \mathbb{R}^{|I|}$ is a function that selects certain dimension(s) of an n -dimensional vector, where the index set I used for selection is a nonempty subset of the whole index set $\{1, \dots, n\}$. For instance, $w_{\{1,4\}}(x) = (x_1, x_4)$ for any $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. We use W to denote the set of all possible selection windows for \mathbb{R}^n , that is,

$$W := \{w_I \mid I \subseteq \{1, \dots, n\}, I \neq \emptyset\}. \quad (9.1)$$

A *basis descriptor* is an atomic arithmetic operation. In this thesis, we restrict our attention to a combinatorial setting, considering primitive arithmetic operations for integers. For instance, `sort`, `order` (a fine-tuned `argsort`), `diff`, and/or `modm` for various $m \in \mathbb{Z}$. We use B to denote a set of pre-selected basis descriptors, from which we further define the depth- k descriptor pool by

$$D^{[k]} := \{b_{k'} \circ \dots \circ b_1 \mid b_{k'}, \dots, b_1 \in B, 0 \leq k' \leq k\}, \quad (9.2)$$

where by convention the composition $b_{k'} \circ \dots \circ b_1 = \text{id}$ if $k' = 0$. So, every descriptor $d \in D^{[k]}$

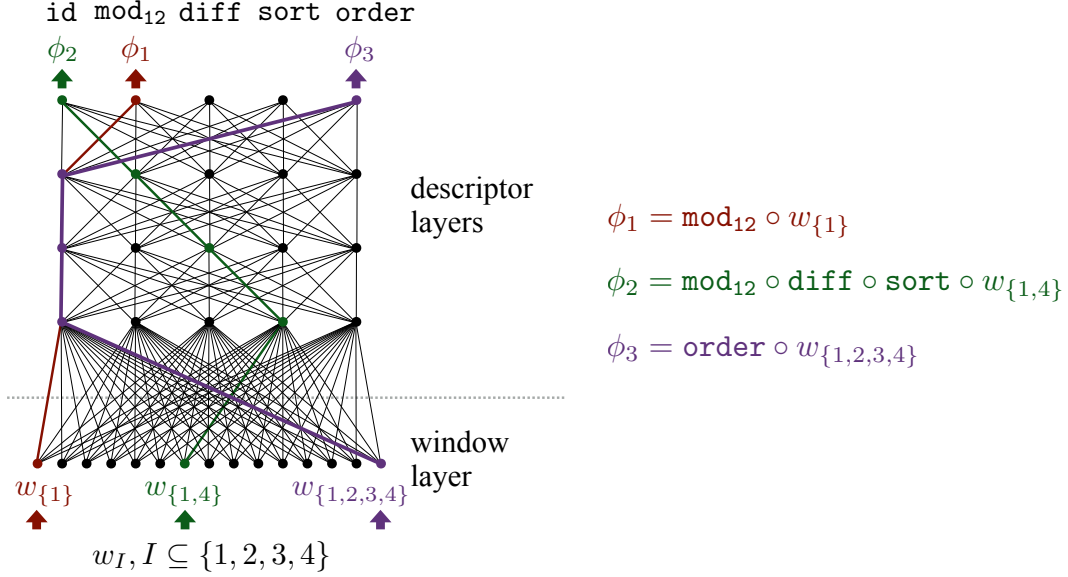


Figure 9.1: An example of feature generation from windows and descriptors.

is a composition of at most k basis descriptors.

Combining windows and descriptors sequentially, we define the depth- $(k + 1)$ feature pool by the following form:

$$\Phi^{[k+1]} := D^{[k]} \circ W := \{d \circ w \mid d \in D^{[k]}, w \in W\}. \quad (9.3)$$

Figure 9.1, from an equivalent view of (9.3), illustrates one example of feature generation from windows and descriptors, where the input data space $X = \mathbb{R}^4$, the basis descriptors $B = \{\text{mod}_{12}, \text{diff}, \text{sort}, \text{order}\}$, and the depth $k = 4$.

The fact that every candidate feature function $\phi \in \Phi^{[k+1]}$ is systematically constructed as a composition of finitely many (here at most k) basis descriptors and a selection window ensures its interpretability. We can step by step read out the meaning of a feature function backwards from its composition formula: for every data point $x \in X$, $\phi(x) = d \circ w(x)$ literally tells us where to look (specified by $w \in W$) and what to look for (specified by $d \in D^{[k]}$).

Now to systematically generate abstractions from features, all we need to pre-specify is a set of basis descriptors B and a depth k (W is automatically determined by the dimensionality of the input data space). Given B and k , we construct a family of feature-induced abstractions accordingly to the following definition:

$$\mathfrak{P}_X^{\Phi^{[k+1]}} := \{\mathcal{P}_\phi \mid \phi \in \Phi^{[k+1]}\}. \quad (9.4)$$

Note that $\mathfrak{P}_X^{\Phi^{[k+1]}}$ is not an abstraction (semi)universe yet. To complete the hierarchy of any abstraction family regardless of its generating mechanisms (i.e. no matter it is induced from features or symmetries or both or something else), there is a generic algorithm [78] that can be used to determine the coarser than relation (i.e. the partial order) between every pair of partitions in the family. However, to increase the efficiency, there are various heuristics we can leverage, including the generic ones and the ones specific to feature-induced abstractions as well as the ones specific to the pre-selected basis descriptors in feature-induced abstractions. We briefly mention two major types of heuristics below for completing a feature-induced abstraction hierarchy.

H1. Same-window heuristic. For any $\phi, \phi' \in \Phi^{[k+1]}$ where $\phi = d \circ w_I, \phi' = d' \circ w_I$:
if there exists some $d'' \in D^{[k]}$ such that $d = d'' \circ d'$, then $\mathcal{P}_\phi \preceq \mathcal{P}_{\phi'}$.

H2. Same-descriptor heuristic. For any $\phi, \phi' \in \Phi^{[k+1]}$ where $\phi = d \circ w_I, \phi' = d \circ w_{I'}$:
if $I \subseteq I'$ and d comprises all coordinate-wise operations, then $\mathcal{P}_\phi \preceq \mathcal{P}_{\phi'}$.

The validity of the above two heuristics is an easy check, and one derived heuristic from combining the two is immediate when neither the windows nor the descriptors are the same but both satisfy all the conditions in H1 and H2. Of course, whenever possible, we can always use a generic heuristic regarding the transitivity property of a partial order, i.e. $\mathcal{P}_\phi \preceq \mathcal{P}_{\phi'}$ and $\mathcal{P}_{\phi'} \preceq \mathcal{P}_{\phi''} \implies \mathcal{P}_\phi \preceq \mathcal{P}_{\phi''}$. In Part III, via a specific case study in music, we will present a detailed implementation of the above algorithmic principles for feature-induced abstractions, including the design choices of basis descriptors and the applications of various heuristics for hierarchy completion.

9.2 SYMMETRY GENERATION: TOP-DOWN APPROACH

Regarding symmetry-induced abstractions, we consider a subgroup lattice with each subgroup in the lattice acting on the same data space X . Conceptually, the complete subgroup lattice $\mathcal{H}_{F(X)}^*$ for the whole transformation group $F(X)$ comprises all possible symmetries. However, by Theorem 6.2, $|\mathcal{H}_{F(X)}^*| \geq |\mathfrak{P}_X^*| = B_{|X|}$, i.e. no less than the $|X|$ -th Bell number. This suggests that even for small X , the complete subgroup lattice can be too large for any realistic computational methods. This section presents the top-down approach to sample a finite subgroup sublattice from the complete lattice $\mathcal{H}_{F(X)}^*$, so that algorithmic operations on this finite sample, including the sampling process itself, is computationally feasible.

We start with the transformation group of $X = \mathbb{R}^n$, and the plan is to consider special subgroups of $F(\mathbb{R}^n)$ and special subspaces of \mathbb{R}^n . In order to systematically enumerate

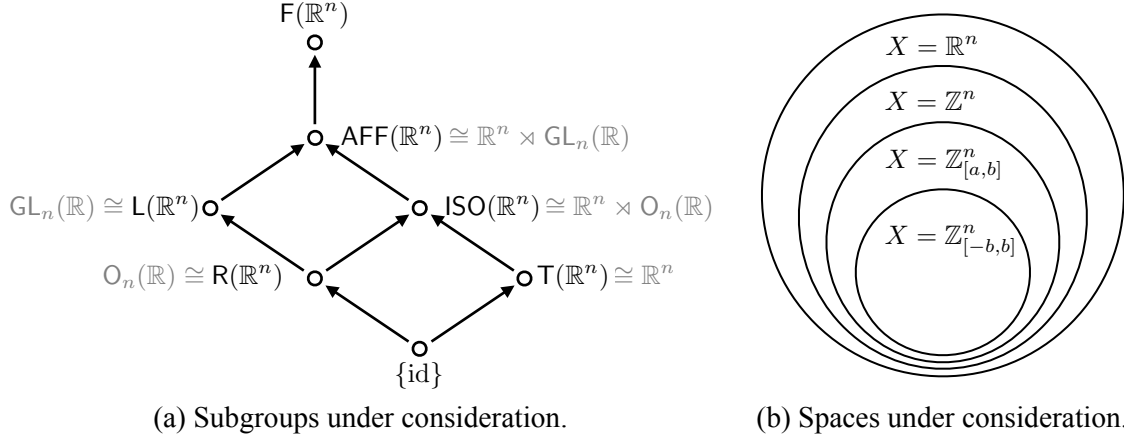


Figure 9.2: Special subgroups and spaces as well as their hierarchies. (a) presents a backbone of the complete subgroup lattice $\mathcal{H}_{\mathbb{F}(\mathbb{R}^n)}^*$, including important subgroups and their breakdowns. One can check the above directed acyclic graph indeed represents a sublattice: it is closed under both join and meet. (b) presents important subspaces of \mathbb{R}^n , where restrictions are gradually added to eventually lead to practical abstraction-construction algorithms.

subgroups of a chosen special subgroup, we derive a principle that allows us to hierarchically break the enumeration problem into smaller and smaller enumeration subproblems. This hierarchical breakdown can guide us in restricting both the type of subgroups and the type of subspaces, so that the resulting abstraction (semi)universe fits our desiderata, and more importantly can be computed in practice. Figure 9.2 presents an outline consisting of special subgroups and subspaces considered in this section as well as their hierarchies.

The central theorem (Theorem 9.5) in this section is a complete identification of every subgroup of the affine transformation group, denoted $\text{AFF}(\mathbb{R}^n)$, from which the identification of smaller affine subgroups can systematically be parametrized. We will take a few steps to get there, starting from characterizing the affine transformation group itself.

Keywords and notations: group homomorphism, isomorphism (\cong); normalizer of a set in a group ($N_G(S) := \{g \in G \mid gSg^{-1} = S\}$), normal subgroup (\trianglelefteq); group decomposition, inner semi-direct product, outer semi-direct product (\rtimes).

9.2.1 The Affine Transformation Group $\text{AFF}(\mathbb{R}^n)$

An *affine transformation* of \mathbb{R}^n is a function $f_{A,u} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ of the form

$$f_{A,u}(x) = Ax + u \quad \text{for any } x \in \mathbb{R}^n, \quad (9.5)$$

where $A \in \text{GL}_n(\mathbb{R})$ is an $n \times n$ real invertible matrix and $u \in \mathbb{R}^n$ is an n -dimensional real vector. We use $\text{AFF}(\mathbb{R}^n)$ to denote the set of all affine transformations of \mathbb{R}^n . There are two special cases:

1. A *translation* of \mathbb{R}^n is a function $t_u : \mathbb{R}^n \rightarrow \mathbb{R}^n$ of the form $x \mapsto x + u$ where $u \in \mathbb{R}^n$; we use $\text{T}(\mathbb{R}^n)$ to denote the set of all translations of \mathbb{R}^n .
2. A *linear transformation* of \mathbb{R}^n is a function $r_A : \mathbb{R}^n \rightarrow \mathbb{R}^n$ of the form $x \mapsto Ax$ where $A \in \text{GL}_n(\mathbb{R})$; we use $\text{L}(\mathbb{R}^n)$ to denote the set of all linear transformations of \mathbb{R}^n .

It is easy to check that $\text{T}(\mathbb{R}^n), \text{L}(\mathbb{R}^n) \leq \text{AFF}(\mathbb{R}^n) \leq \text{F}(\mathbb{R}^n)$; further, $(\text{T}(\mathbb{R}^n), \circ)$ and $(\text{L}(\mathbb{R}^n), \circ)$ are isomorphic to $(\mathbb{R}^n, +)$ and $(\text{GL}_n(\mathbb{R}), \cdot)$, respectively. It is known that

$$\text{AFF}(\mathbb{R}^n) = \text{T}(\mathbb{R}^n) \circ \text{L}(\mathbb{R}^n) \cong \text{T}(\mathbb{R}^n) \rtimes \text{L}(\mathbb{R}^n) \cong \mathbb{R}^n \rtimes \text{GL}_n(\mathbb{R}). \quad (9.6)$$

So every affine transformation can be uniquely identified with a pair $(u, A) \in \mathbb{R}^n \rtimes \text{GL}_n(\mathbb{R})$. In particular, the identity transformation is identified with $(\mathbf{0}, I)$, the translation group $\text{T}(\mathbb{R}^n)$ is identified with $\{(u, I) \mid u \in \mathbb{R}^n\}$, and the linear transformation group $\text{L}(\mathbb{R}^n)$ is identified with $\{(\mathbf{0}, A) \mid A \in \text{GL}_n(\mathbb{R})\}$. Under this identification, compositions and inverses of affine transformations become

$$(u, A)(u', A') = (u + Au', AA') \quad \text{and} \quad (u, A)^{-1} = (-A^{-1}u, A^{-1}). \quad (9.7)$$

The above identification further allows us to introduce two functions $\ell : \text{AFF}(\mathbb{R}^n) \rightarrow \text{GL}_n(\mathbb{R})$ and $\tau : \text{AFF}(\mathbb{R}^n) \rightarrow \mathbb{R}^n$ to extract the linear and translation part of an affine transformation, respectively, where

$$\ell(f_{A,u}) = A, \quad \tau(f_{A,u}) = u \quad \text{for any } f_{A,u} \in \text{AFF}(\mathbb{R}^n). \quad (9.8)$$

Now we can start our journey towards a complete identification of every subgroup H of $\text{AFF}(\mathbb{R}^n)$. We introduce the first foundational quantity $T := \text{T}(\mathbb{R}^n) \cap H$, which is the set of pure translations in H , called the *translation subgroup* of H . It is easy to check that $T \trianglelefteq H$ since translations are normal in affine transformations. Therefore, the quotient group $H/T = \{T \circ h \mid h \in H\}$ is well-defined. The elements in H/T are called *cosets*. The following theorems reveal more structures of H/T , the second foundational quantity.

Lemma 9.1. $\ell : \text{AFF}(\mathbb{R}^n) \rightarrow \text{GL}_n(\mathbb{R})$ is a homomorphism.

Proof. For any $f_{A,u}, f_{A',u'} \in \text{AFF}(\mathbb{R}^n)$, we have $\ell(f_{A,u} \circ f_{A',u'}) = \ell(f_{AA', Au'+u}) = AA' = \ell(f_{A,u})\ell(f_{A',u'})$, which implies that ℓ is a homomorphism.

Theorem 9.1. *Let $H \leq \text{AFF}(\mathbb{R}^n)$, $T = \mathbb{T}(\mathbb{R}^n) \cap H$. Then $h, h' \in H$ are in the same coset in H/T if and only if they have the same linear part, i.e. $\ell(h) = \ell(h')$.*

Proof. It is straightforward to check that

$$T \circ h = T \circ h' \iff h' \circ h^{-1} \in T \iff \ell(h' \circ h^{-1}) = I \iff \ell(h') = \ell(h). \quad (9.9)$$

The last if-and-only-if condition holds because $\ell(h' \circ h^{-1}) = \ell(h')\ell(h)^{-1}$ by Lemma 9.1.

Theorem 9.2. *Let $H \leq \text{AFF}(\mathbb{R}^n)$, $T = \mathbb{T}(\mathbb{R}^n) \cap H$. If $h, h' \in H$ are in the same coset in H/T , then $\tau(h') - \tau(h) \in \tau(T) := \{u \mid t_u \in T\}$.*

Proof. Let $h, h' \in H$ be any two affine transformations in the same coset in H/T , then this means $T \circ h = T \circ h'$, or equivalently $h' \circ h^{-1} \in T$. By (9.7), we have

$$\tau(h' \circ h^{-1}) = \tau(h') + \ell(h')\tau(h^{-1}) = \tau(h') + \ell(h')(-\ell(h)^{-1}\tau(h)) = \tau(h') - \tau(h), \quad (9.10)$$

where the last equality holds by Theorem 9.1. Therefore, $\tau(h') - \tau(h) \in \tau(T)$.

Remark 9.1. *Theorems 9.1 and 9.2 present two characterizations of elements in the same coset in H/T . The former, through the linear part, is an if-and-only-if characterization; while the latter, through the translation part, is a necessary but not sufficient characterization.*

Theorem 9.3. *Let $H \leq \text{AFF}(\mathbb{R}^n)$, $T = \mathbb{T}(\mathbb{R}^n) \cap H$. Then $H/T \cong \ell(H)$.*

Proof. It is clear that $\ell(H) \leq \text{GL}_n(\mathbb{R})$, since $H \leq \text{AFF}(\mathbb{R}^n)$ and ℓ is a homomorphism (Lemma 9.1) which preserves subgroups. Let $\bar{\ell} : H/T \rightarrow \ell(H)$ be the function of the form $\bar{\ell}(T \circ h) = \ell(h)$, we claim that $\bar{\ell}$ is an isomorphism. To see this, for any $T \circ h, T \circ h' \in H/T$,

$$\bar{\ell}((T \circ h)(T \circ h')) = \bar{\ell}(T \circ (h \circ h')) = \ell(h \circ h') = \ell(h)\ell(h') = \bar{\ell}(T \circ h)\bar{\ell}(T \circ h'), \quad (9.11)$$

which implies $\bar{\ell}$ is a homomorphism. Further, for any $T \circ h, T \circ h' \in H/T$, if $\bar{\ell}(T \circ h) = \bar{\ell}(T \circ h')$, then $\ell(h) = \ell(h')$. By Theorem 9.1, this implies that $T \circ h = T \circ h'$, so $\bar{\ell}$ is injective. Lastly, for any $A \in \ell(H)$, there exists an $h \in H$ such that $\ell(h) = A$. For this particular h , $T \circ h \in H/T$, and $\bar{\ell}(T \circ h) = \ell(h) = A$. This implies that $\bar{\ell}$ is surjective. Therefore, $\bar{\ell}$ is a bijective homomorphism, i.e. an isomorphism.

Remark 9.2. *Theorem 9.3 can be proved directly from the first isomorphism theorem, by recognizing $\ell|_H$ is a homomorphism whose kernel and image are T and $\ell(H)$, respectively. However, the above proof explicitly gives the isomorphism $\bar{\ell}$ which is useful in the sequel.*

Theorem 9.4 (Compatibility). *Let $H \leq \text{AFF}(\mathbb{R}^n)$, $T = \mathbb{T}(\mathbb{R}^n) \cap H$. For any $A \in \ell(H)$ and $v \in \tau(T)$, we have $Av \in \tau(T)$. Further, if we define a function $\cdot : \ell(H) \times \tau(T) \rightarrow \tau(T)$ of the form $(A, v) \mapsto Av$, then \cdot is a group action of $\ell(H)$ on $\tau(T)$.*

Proof. For any $A \in \ell(H)$ and $v \in \tau(T)$, there exists an $f_{A,u} \in H$ and an $f_{I,v} \in T$. Since $T \trianglelefteq H$, then $f_{A,u} \circ f_{I,v} \circ f_{A,u}^{-1} \in T$. By (9.7) we have that $f_{A,u} \circ f_{I,v} \circ f_{A,u}^{-1} = f_{I,Av}$, so $f_{I,Av} \in T$, i.e. $Av = \tau(f_{I,Av}) \in \tau(T)$. To see $\cdot : \ell(H) \times \tau(T) \rightarrow \tau(T)$ defines a group action of $\ell(H)$ on $\tau(T)$ is then easy, since it is a matrix-vector multiplication. A quick check shows that for any $v \in \tau(T)$, $I \cdot v = v$; for any $v \in \tau(T)$ and $A, B \in \ell(H)$, $A \cdot (B \cdot v) = (AB) \cdot v$.

So far, we have seen that for any subgroup $H \leq \text{AFF}(\mathbb{R}^n)$, its subset of pure translations $T := \mathbb{T}(\mathbb{R}^n) \cap H$ is a normal subgroup of H ; T is also a normal subgroup of $\mathbb{T}(\mathbb{R}^n)$, since $\mathbb{T}(\mathbb{R}^n)$ is a commutative group. As a result, both quotient groups H/T and $\mathbb{T}(\mathbb{R}^n)/T$ are well-defined. We next introduce a function, called a *vector system*, which connects the two quotient groups. It turns out that vector systems comprise the last piece of information that leads to a complete identification of every subgroup of $\text{AFF}(\mathbb{R}^n)$. Note that $H/T \cong \ell(H)$ (Theorem 9.3) and $\mathbb{T}(\mathbb{R}^n)/T \cong \mathbb{R}^n/\tau(T)$; thus for conceptual ease (think in terms of matrices and vectors), we introduce vector systems connecting $\ell(H)$ and $\mathbb{R}^n/\tau(T)$ instead.

Definition 9.1 (Vector system). *For any $L \leq \text{GL}_n(\mathbb{R})$ and $V \leq \mathbb{R}^n$, an (L, V) -vector system is a function $\xi : L \rightarrow \mathbb{R}^n/V$, which in addition satisfies the following two conditions:*

1. *compatibility condition: for any $A \in L$, $AV = \{Av \mid v \in V\} = V$;*
2. *cocycle condition: for any $A, A' \in L$, $\xi(AA') = \xi(A) + A\xi(A')$.*

Note: elements in \mathbb{R}^n/V are cosets of the form $V + u$ for $u \in \mathbb{R}^n$. It is easy to check: for any two cosets in \mathbb{R}^n/V , the sum

$$(V + u) + (V + u') = \{v + u + v' + u' \mid v, v' \in V\} = V + (u + u'); \quad (9.12)$$

for any $A \in L$ and any coset in \mathbb{R}^n/V , the product

$$A(V + u) = \{A(v + u) \mid v \in V\} = V + Au. \quad (9.13)$$

So, the sum and product in the cocycle condition are defined in the above sense.

We use $\Xi_{L,V}$ to denote the family of all (L, V) -vector systems. One can check that $\Xi_{L,V} \neq \emptyset$ if and only if L, V are compatible (consider the trivial vector system $\xi_{L,V}^0$ given by $\xi_{L,V}^0(A) = V$ for all $A \in L$). We use $\Xi^* := \{\Xi_{L,V} \mid L \leq \text{GL}_n(\mathbb{R}), V \leq \mathbb{R}^n \text{ compatible}\}$ to denote the universe of all vector systems.

Remark 9.3. *The universe of all vector systems Ξ^* can be parameterized by the set of compatible pairs $(L, V) \in \mathcal{H}_{\text{GL}_n(\mathbb{R})}^* \times \mathcal{H}_{\mathbb{R}^n}^*$. The reason is straightforward: L and V respectively define the domain and codomain of a function, and two functions are different if either their domains or their codomains are different.*

Lemma 9.2. *Let $L \leq \text{GL}_n(\mathbb{R})$, $V \leq \mathbb{R}^n$, and $\xi \in \Xi_{L,V}$, then*

1. *for the identity matrix $I \in L$, $\xi(I) = V$;*

2. *for any $A \in L$, $\xi(A^{-1}) = -A^{-1}\xi(A)$.*

Proof. For any $A \in L$, $\xi(A) = \xi(IA) = \xi(I) + I\xi(A) = \xi(I) + \xi(A)$. Note that $\xi(A), \xi(I) \in \mathbb{R}^n/V$, so $\xi(A) = V + a$ and $\xi(I) = V + b$ for some $a, b \in \mathbb{R}^n$. Thus,

$$\xi(A) = \xi(I) + \xi(A) \implies V + a = V + (b + a). \quad (9.14)$$

This further implies that $b \in V$ and $\xi(I) = V + b = V$.

For any $A \in L$, $V = \xi(A^{-1}A) = \xi(A^{-1}) + A^{-1}\xi(A)$. Note that $\xi(A), \xi(A^{-1}) \in \mathbb{R}^n/V$, so $\xi(A) = V + a$ and $\xi(A^{-1}) = V + c$ for some $a, c \in \mathbb{R}^n$. Thus,

$$V = \xi(A^{-1}) + A^{-1}\xi(A) \implies V = V + (c + A^{-1}a). \quad (9.15)$$

This further implies that $c + A^{-1}a \in V$, or equivalently, $c \in V + (-A^{-1}a)$. Therefore, c and $-A^{-1}a$ are in the same coset and $\xi(A^{-1}) = V + c = V + (-A^{-1}a) = -A^{-1}\xi(A)$.

Theorem 9.5 (Affine subgroup identification). *Let*

$$\Sigma := \{(L, V, \xi) \mid L \leq \text{GL}_n(\mathbb{R}), V \leq \mathbb{R}^n, \xi \in \Xi_{L,V}\}, \quad (9.16)$$

then there is a bijection between $\mathcal{H}_{\text{AFF}(\mathbb{R}^n)}^$ and Σ .*

Proof. Let $\Psi : \mathcal{H}_{\text{AFF}(\mathbb{R}^n)}^* \rightarrow \Sigma$ be the function defined by

$$\Psi(H) := (\ell(H), \tau(T), \xi_H) \quad \text{for any } H \in \mathcal{H}_{\text{AFF}(\mathbb{R}^n)}^*, \quad (9.17)$$

where $T := \mathbb{T}(\mathbb{R}^n) \cap H$, and $\xi_H : \ell(H) \rightarrow \mathbb{R}^n/\tau(T)$ is given by $\xi_H(A) = \tau(\bar{\ell}^{-1}(A))$ with $\bar{\ell} : H/T \rightarrow \ell(H)$ being the isomorphism defined in the proof of Theorem 9.3. We first show Ψ is well-defined, and then show it is bijective; in particular, we will show that the inverse function has an exact formula as follows:

$$\Psi^{-1}((L, V, \xi)) = \{f_{A,u} \in \text{AFF}(\mathbb{R}^n) \mid A \in L, u \in \xi(A)\} \quad \text{for any } (L, V, \xi) \in \Sigma. \quad (9.18)$$

The entire proof is divided into four parts.

1. Check that ξ_H is well-defined. More specifically, we want to show that

$$\xi_H(A) \in \mathbb{R}^n/\tau(T) \quad \text{for any } H \in \mathcal{H}_{\text{AFF}}^*(\mathbb{R}^n) \text{ and } A \in \ell(H). \quad (9.19)$$

For any $A \in \ell(H)$, $\bar{\ell}^{-1}(A)$ is the coset $T \circ h$ in H/T such that $\ell(h) = A$. Pick any $h \in \bar{\ell}^{-1}(A)$ which is possible since as a coset $\bar{\ell}^{-1}(A) \neq \emptyset$. For any $h' \in \bar{\ell}^{-1}(A)$, by Theorem 9.2, $\tau(h') - \tau(h) \in \tau(T)$, i.e. $\tau(h') \in \tau(T) + \tau(h)$, so $\tau(\bar{\ell}^{-1}(A)) \subseteq \tau(T) + \tau(h)$. Conversely, for any $w \in \tau(T) + \tau(h)$, there exists a $v \in \tau(T)$ such that $w = v + \tau(h)$. Note that the pure translation $t_v \in T \leq H$ and $h \in \bar{\ell}^{-1}(A) \subseteq H$, so their composition $t_v \circ h \in H$. Further, it is an easy check that $\ell(t_v \circ h) = A$ and $\tau(t_v \circ h) = v + \tau(h) = w$. This implies that we have found $h' := t_v \circ h \in \bar{\ell}^{-1}(A)$ and $\tau(h') = w$, thus, $w \in \tau(\bar{\ell}^{-1}(A))$. This finally yields that $\tau(T) + \tau(h) \subseteq \tau(\bar{\ell}^{-1}(A))$. Combining the two directions, we have $\tau(\bar{\ell}^{-1}(A)) = \tau(T) + \tau(h)$; so, $\xi_H(A) = \tau(\bar{\ell}^{-1}(A)) \in \mathbb{R}^n/\tau(T)$. This implies ξ_H is well-defined.

2. Check that Ψ is well-defined. More specifically, we want to show that

$$\Psi(H) \in \Sigma \quad \text{for any } H \in \mathcal{H}_{\text{AFF}}^*(\mathbb{R}^n). \quad (9.20)$$

For any $H \in \mathcal{H}_{\text{AFF}}^*(\mathbb{R}^n)$, it is clear that $\ell(H) \leq \text{GL}_n(\mathbb{R})$, $\tau(T) \leq \mathbb{R}^n$, and they are compatible (Theorem 9.4); therefore, it suffices to show that $\xi_H \in \Xi_{\ell(H), \tau(T)}$. Note that, for any $A, A' \in \ell(H)$, the product of two cosets $\bar{\ell}^{-1}(A)\bar{\ell}^{-1}(A') = (T \circ f_{A,u})(T \circ f_{A',u'}) = T \circ (f_{A,u} \circ f_{A',u'}) = T \circ f_{AA',u+Au'}$, for some $f_{A,u}, f_{A',u'} \in H$. Therefore,

$$\xi_H(AA') = \tau(\bar{\ell}^{-1}(AA')) = \tau(\bar{\ell}^{-1}(A)\bar{\ell}^{-1}(A')) = \tau(T \circ f_{AA',u+Au'}) = \tau(T) + u + Au'. \quad (9.21)$$

On the other hand,

$$\xi_H(A) + A\xi_H(A') = (\tau(T) + u) + A(\tau(T) + u') = \tau(T) + u + Au'. \quad (9.22)$$

Therefore, $\xi_H(AA') = \xi_H(A) + A\xi_H(A')$ and $\xi_H \in \Xi_{\ell(H), \tau(T)}$. This implies that for any $H \in \mathcal{H}_{\text{AFF}}^*(\mathbb{R}^n)$, $\Psi(H) \in \Sigma$, so Ψ is well-defined.

3. Check that Ψ is injective. Pick any $H, H' \in \mathcal{H}_{\text{AFF}}^*(\mathbb{R}^n)$ and suppose $\Psi(H) = \Psi(H')$, i.e. $(\ell(H), \tau(T), \xi_H) = (\ell(H'), \tau(T'), \xi_{H'})$, where $T := \mathbb{T}(\mathbb{R}^n) \cap H$ and $T' := \mathbb{T}(\mathbb{R}^n) \cap H'$. For any $f_{A,u} \in H$, $A = \ell(f_{A,u}) \in \ell(H) = \ell(H')$; thus, there exists some $f_{A,u'} \in H'$. Let $\bar{\ell} : H/T \rightarrow \ell(H)$ and $\bar{\ell}' : H'/T' \rightarrow \ell(H')$ be the isomorphisms similarly defined as in

Theorem 9.3. As proved earlier, we have

$$\xi_H(A) = \tau(\bar{\ell}^{-1}(A)) = \tau(T) + \tau(f_{A,u}) = \tau(T) + u, \quad (9.23)$$

$$\xi_{H'}(A) = \tau(\bar{\ell}'^{-1}(A)) = \tau(T') + \tau(f_{A,u'}) = \tau(T) + u'. \quad (9.24)$$

Therefore, $\tau(T) + u = \tau(T) + u'$. This implies that $\tau(f_{A,u}) = u \in \tau(T) + u' = \tau(\bar{\ell}'^{-1}(A))$. So, $f_{A,u} \in \bar{\ell}'^{-1}(A) \subseteq H'$, and $H \subseteq H'$. By a completely symmetrical process, $H' \subseteq H$. Therefore, $H = H'$, which implies that Ψ is injective.

4. Check that Ψ is surjective. Pick any $(L, V, \xi) \in \Sigma$ and let

$$H := \{f_{A,u} \in \mathbf{AFF}(\mathbb{R}^n) \mid A \in L, u \in \xi(A)\}. \quad (9.25)$$

We first show that $H \leq \mathbf{AFF}(\mathbb{R}^n)$ by a subgroup test. It is clear $H \subseteq \mathbf{AFF}(\mathbb{R}^n)$. The identity matrix $I \in L$ and $\mathbf{0} \in V = \xi(I)$, so the identity transformation $\text{id} = f_{I,\mathbf{0}} \in H$. For any $f_{A,u}, f_{A',u'} \in H$, we have $A, A' \in L$ and $u \in \xi(A), u' \in \xi(A')$, which respectively implies that $AA' \in L$ and $u + Au' \in \xi(A) + A\xi(A') = \xi(AA')$. So, $f_{A,u} \circ f_{A',u'} = f_{AA',u+Au'} \in H$. For any $f_{A,u} \in H$, we have $A \in L$ and $u \in \xi(A)$, which respectively implies that $A^{-1} \in L$ and $-A^{-1}u \in -A^{-1}\xi(A) = \xi(A^{-1})$. So, $f_{A,u}^{-1} = f_{A^{-1},-A^{-1}u} \in H$. Therefore, $H \leq \mathbf{AFF}(\mathbb{R}^n)$. Now we show that $\Psi(H) = (\ell(H), \tau(T), \xi_H) = (L, V, \xi)$. First, for any $A \in \ell(H)$, there exists an $f_{A,u} \in H$, so $A \in L$ which implies that $\ell(H) \subseteq L$. Conversely, for any $A \in L$, $\xi(A)$ is a coset in \mathbb{R}^n/V , so $\xi(A) \neq \emptyset$. Pick any $u \in \xi(A)$, then $f_{A,u} \in H$, so $A = \ell(f_{A,u}) \in \ell(H)$ which implies $L \subseteq \ell(H)$. Combining both directions yields $\ell(H) = L$. Second, note that $T = \mathbf{T}(\mathbb{R}^n) \cap H = \{f_{I,u} \in \mathbf{AFF}(\mathbb{R}^n) \mid u \in \xi(I) = V\}$, so $\tau(T) = \{u \mid u \in V\} = V$. Third, note that $\xi_H : \ell(H) \rightarrow \mathbb{R}^n/\tau(T)$ and $\xi : L \rightarrow \mathbb{R}^n/V$. We have shown that $\ell(H) = L$ and $\tau(T) = V$, so ξ_H and ξ have the same domain and codomain. Further, for any $A \in L$, $\xi_H(A) = \tau(\bar{\ell}^{-1}(A)) = \tau(\{f_{A,u} \in \mathbf{AFF}(\mathbb{R}^n) \mid u \in \xi(A)\}) = \{u \mid u \in \xi(A)\} = \xi(A)$. So, $\xi_H = \xi$. Now we have $\Psi(H) = (L, V, \xi)$. Therefore, Ψ is surjective. In particular,

$$\Psi^{-1}((L, V, \xi)) = \{f_{A,u} \in \mathbf{AFF}(\mathbb{R}^n) \mid A \in L, u \in \xi(A)\} \quad \text{for any } (L, V, \xi) \in \Sigma. \quad (9.26)$$

Remark 9.4. The bijection Ψ from $\mathcal{H}_{\mathbf{AFF}(\mathbb{R}^n)}^*$ to Σ allows us to use the latter to parameterize the former. Further, through the inverse function Ψ^{-1} , we can enumerate affine subgroups by enumerating triplets $(L, V, \xi) \in \Sigma$, or more specifically, by enumerating matrix subgroups of $\mathbf{GL}_n(\mathbb{R})$, vector subgroups of \mathbb{R}^n , and then vector systems for every compatible pair of a matrix subgroup and a vector subgroup. Note that enumeration for each element in the triplet is still

not practical if no restriction is imposed. Nevertheless, we have broken the original subgroup enumeration problem into three smaller enumeration problems. More importantly, we are now more directed in imposing restrictions on both subgroups and spaces, under which the three smaller enumerations become practical. We will discuss these restrictions (e.g. being isometric, finite, discrete, compact) in more details in the sequel.

9.2.2 The Isometry Group $\text{ISO}(\mathbb{R}^n)$

One way to restrict $\Sigma := \{(L, V, \xi) \mid L \leq \text{GL}_n(\mathbb{R}), V \leq \mathbb{R}^n, \xi \in \Xi_{L,V}\}$ is to consider a special subgroup of $\text{GL}_n(\mathbb{R})$. Instead of all subgroups of $\text{GL}_n(\mathbb{R})$, we consider only subgroups consisting of *orthogonal matrices*. This restriction gives rise to the subgroup lattice $\mathcal{H}_{\text{ISO}(\mathbb{R}^n)}^*$ where $\text{ISO}(\mathbb{R}^n)$ denotes the group of isometries of \mathbb{R}^n . In this subsection, we first give an overview of $\text{ISO}(\mathbb{R}^n)$, and then cast $\mathcal{H}_{\text{ISO}(\mathbb{R}^n)}^*$ in the big picture of $\mathcal{H}_{\text{AFF}(\mathbb{R}^n)}^*$ and Σ .

An *isometry* of \mathbb{R}^n , with respect to the Euclidean distance d , is a transformation $h : \mathbb{R}^n \rightarrow \mathbb{R}^n$ which preserves distances: $d(h(x), h(x')) = d(x, x')$, for all $x, x' \in \mathbb{R}^n$. We use $\text{ISO}(\mathbb{R}^n)$ to denote the set of all isometries of \mathbb{R}^n , which is a subgroup of the transformation group $\text{F}(\mathbb{R}^n)$. So, we call $\text{ISO}(\mathbb{R}^n)$ the *isometry group* of \mathbb{R}^n .

A (*generalized*) *rotation* of \mathbb{R}^n is a linear transformation $r_A : \mathbb{R}^n \rightarrow \mathbb{R}^n$ given by $x \mapsto Ax$, for some orthogonal matrix $A \in \text{O}_n(\mathbb{R}) := \{A \in \mathbb{R}^{n \times n} \mid A^\top = A^{-1}\} \leq \text{GL}_n(\mathbb{R})$. We use $\text{R}(\mathbb{R}^n)$ to denote the set of all rotations of \mathbb{R}^n , which is a subgroup of the linear transformation group $\text{L}(\mathbb{R}^n)$. So, we call $\text{R}(\mathbb{R}^n)$ the *rotation group* of \mathbb{R}^n .

There are two key characterizations of $\text{ISO}(\mathbb{R}^n)$. The first one regards its components:

$$\text{ISO}(\mathbb{R}^n) = \langle \text{T}(\mathbb{R}^n) \cup \text{R}(\mathbb{R}^n) \rangle \quad \text{where } \text{T}(\mathbb{R}^n) \cap \text{R}(\mathbb{R}^n) = \{\text{id}\}. \quad (9.27)$$

This characterization says that $\text{ISO}(\mathbb{R}^n)$ comprises exclusively translations, rotations, and their finite compositions. Note that we can rewrite the above characterization as $\text{T}(\mathbb{R}^n) \vee \text{R}(\mathbb{R}^n) = \text{ISO}(\mathbb{R}^n)$ and $\text{T}(\mathbb{R}^n) \wedge \text{R}(\mathbb{R}^n) = \{\text{id}\}$. This determines the positions of the four subgroups $\{\text{id}\}$, $\text{T}(\mathbb{R}^n)$, $\text{R}(\mathbb{R}^n)$, and $\text{ISO}(\mathbb{R}^n)$ in the subgroup lattice $(\mathcal{H}_{\text{F}(\mathbb{R}^n)}^*, \leq)$, which forms a diamond shape in the direct acyclic graph in Figure 9.2a. The second characterization of $\text{ISO}(\mathbb{R}^n)$ regards a unique representation for every isometry of \mathbb{R}^n , which is done by a group decomposition of $\text{ISO}(\mathbb{R}^n)$ as semi-direct products:

$$\text{ISO}(\mathbb{R}^n) = \text{T}(\mathbb{R}^n) \circ \text{R}(\mathbb{R}^n) \cong \mathbb{R}^n \rtimes \text{O}_n(\mathbb{R}). \quad (9.28)$$

This characterization says that every isometry of \mathbb{R}^n can be uniquely represented as an affine

transformation $f_{A,u} \in \text{AFF}(\mathbb{R}^n)$ where $A \in \text{O}_n(\mathbb{R})$ and $u \in \mathbb{R}^n$. This further implies that $\text{ISO}(\mathbb{R}^n)$ is a special subgroup of $\text{AFF}(\mathbb{R}^n)$.

Let $\Psi : \mathcal{H}_{\text{AFF}(\mathbb{R}^n)}^* \rightarrow \Sigma$ be the bijection defined in the proof of Theorem 9.5, and let

$$\Sigma' := \{(L, V, \xi) \mid L \leq \text{O}_n(\mathbb{R}), V \leq \mathbb{R}^n, \xi \in \Xi_{L,V}\} \subseteq \Sigma. \quad (9.29)$$

One can check: $\Psi^{-1}(\Sigma') = \mathcal{H}_{\text{ISO}(\mathbb{R}^n)}^*$. This means $\Psi|_{\mathcal{H}_{\text{ISO}(\mathbb{R}^n)}^*} : \mathcal{H}_{\text{ISO}(\mathbb{R}^n)}^* \rightarrow \Sigma'$ is well-defined and bijective. Therefore, the subgroups of $\text{ISO}(\mathbb{R}^n)$ can be enumerated by the triplets in Σ' in a similar manner as in the remark following Theorem 9.5. The only difference is that we now enumerate subgroups of $\text{O}_n(\mathbb{R})$ instead of the entire $\text{GL}_n(\mathbb{R})$.

Note that restricting to subgroups of $\text{O}_n(\mathbb{R})$ does not really make the enumeration problem practical. However, there are many ways of imposing additional restrictions on $\text{ISO}(\mathbb{R}^n)$ to eventually achieve practical enumerations. We want to point out that there is no universal way of constraining the infinite enumeration problem into a practical one: the design of restrictions is most effective if it is consistent with the underlying topic domain. So, for instance, one can start with his/her intuition to try out some restrictions whose effectivenesses can be verified via a subsequent learning process (Chapter 10). In the next subsection, we give two examples to illustrate some of the existing design choices that have been made in two different domains.

9.2.3 Special subgroups of $\text{ISO}(\mathbb{R}^n)$ used in Chemistry and Music

From two examples, we show how additional restrictions can be imposed to yield a finite collection of subgroups of $\text{ISO}(\mathbb{R}^n)$, capturing different parts of the infinite subgroup lattice $\mathcal{H}_{\text{ISO}(\mathbb{R}^n)}^*$. The two examples are from two different topic domains: one is from chemistry (or more precisely, crystallography), the other is from music. The ways of adding restrictions in these two examples are quite different: one introduces conjugacy relations to obtain a finite collection of subgroup types; the other restricts the space to be discrete or even finite.

The First Example: Crystallographic Space Groups

In crystallography, symmetry is used to characterize crystals, to identify repeating parts of molecules, and to simplify both data collection and subsequent calculations. Further, the symmetry of physical properties of a crystal such as thermal conductivity and optical activity has a strong connection with the symmetry of the crystal. So, a thorough knowledge of symmetry is crucial to a crystallographer. A complete set of symmetry classes is captured

by a collection of 230 unique 3-dimensional space groups. However, space groups represent a special type of subgroups of $\text{ISO}(\mathbb{R}^n)$ which can be defined in general for any dimension.

We give a short review of known results from crystallography, and then identify space groups in the parametrization set Σ that we derived earlier. A *crystallographic space group* or *space group* Γ is a discrete (with respect to the subset topology) and cocompact (i.e. the abstraction space $\pi(\Gamma) := \mathbb{R}^n/\Gamma$ is compact with respect to the quotient topology) subgroup of $\text{ISO}(\mathbb{R}^n)$. So, if the underlying topic domain indeed considers only compact abstractions, space groups are good candidates. A major reason is that for a given dimension, there exist only finitely many space groups (up to isomorphism or affine conjugacy) by Bieberbach's second and third theorems [79, 80].

Bieberbach's first theorem [79, 80] gives an equivalent characterization of space groups: a subgroup Γ of $\text{ISO}(\mathbb{R}^n)$ is a space group if $T := \mathbb{T}(\mathbb{R}^n) \cap \Gamma$ is isomorphic to \mathbb{Z}^n and $\tau(T)$ spans \mathbb{R}^n . In particular, for a space group Γ in standard form, we have $\ell(\Gamma) \leq \mathbf{O}_n(\mathbb{Z})$, $\tau(T) = \mathbb{Z}^n$ [81]. Therefore, we can use

$$\Sigma''_{cryst} := \{(L, V, \xi) \mid L \leq \mathbf{O}_n(\mathbb{Z}), V = \mathbb{Z}^n, \xi \in \Xi_{L, V}\} \subseteq \Sigma' \subseteq \Sigma \quad (9.30)$$

to parameterize the set of all space groups in standard form. We will soon (in the following example) see that $|\mathbf{O}_n(\mathbb{Z})| = n!2^n$ which is finite. For every $L \leq \mathbf{O}_n(\mathbb{Z})$, the enumeration of vector systems $\xi \in \Xi_{L, \mathbb{Z}^n}$ is also made feasible in [82] by identifying orbits in $H^1(L, \mathbb{R}^n/\mathbb{Z}^n)$ under the group action of $\text{N}_{\text{GL}_n(\mathbb{Z})}(L)$ on $H^1(L, \mathbb{R}^n/\mathbb{Z}^n)$, where $H^1(L, \mathbb{R}^n/\mathbb{Z}^n)$ is the first cohomology group of L with values in $\mathbb{R}^n/\mathbb{Z}^n$ and $\text{N}_{\text{GL}_n(\mathbb{Z})}(L)$ is the integral normalizer of L . We refer interested readers to the original Zassenhaus algorithm [82] and the GAP package CrystCat [83] for more details on the algorithmic implementation of space groups.

The Second Example: Isometries of \mathbb{Z}^n in Music

Another example of obtaining a finite collection of subgroups of $\text{ISO}(\mathbb{R}^n)$ comes from computational music theory. This is an extension to our earlier work on building an automatic music theorist [50–53]. In this example, we impose restrictions on the space, focusing on discrete subsets of \mathbb{R}^n that represent music pitches from equal temperament. Restrictions on the space further result in restrictions on the subgroups under consideration, namely only those subgroups that *stabilize* the restricted subsets of \mathbb{R}^n . We start our discussion on isometries of \mathbb{Z}^n , while further restrictions for a finite discrete subspace such as $\mathbb{Z}^n_{[a,b]}$ or $\mathbb{Z}^n_{[-b,b]}$ (Figure 9.2b) will be presented in Section 9.4. We first introduce a few definitions regarding the space \mathbb{Z}^n in parallel with their counterparts regarding \mathbb{R}^n , and then establish

their equivalences under *restricted setwise stabilizers*.

Definition 9.2. An isometry of \mathbb{Z}^n , with respect to the Euclidean distance d (or more precisely $d|_{\mathbb{Z}^n}$) is a function $h' : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ which preserves distances: $d(h'(x), h'(x')) = d(x, x')$, for all $x, x' \in \mathbb{Z}^n$. We use $\text{ISO}(\mathbb{Z}^n)$ to denote the set of all isometries of \mathbb{Z}^n .

Definition 9.3. A translation of \mathbb{Z}^n is a function $t'_u : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ of the form $x \mapsto x + u$, where $u \in \mathbb{Z}^n$. We use $\text{T}(\mathbb{Z}^n)$ to denote the set of all translations of \mathbb{Z}^n .

Definition 9.4. A (generalized) rotation of \mathbb{Z}^n is a function $r'_A : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ of the form $x \mapsto Ax$, where $A \in \text{O}_n(\mathbb{Z}) := \{A \in \mathbb{Z}^{n \times n} \mid A^\top = A^{-1}\}$. We use $\text{R}(\mathbb{Z}^n)$ to denote the set of all rotations of \mathbb{Z}^n .

It is easy to check that $(\text{T}(\mathbb{Z}^n), \circ)$ is isomorphic to $(\mathbb{Z}^n, +)$, and $(\text{R}(\mathbb{Z}^n), \circ)$ is isomorphic to $(\text{O}_n(\mathbb{Z}), \cdot)$; further, $\text{T}(\mathbb{Z}^n), \text{R}(\mathbb{Z}^n) \leq \text{F}(\mathbb{Z}^n)$, and $\text{T}(\mathbb{Z}^n), \text{R}(\mathbb{Z}^n) \subseteq \text{ISO}(\mathbb{Z}^n)$, so translations and rotations of \mathbb{Z}^n are transformations and are also isometries. However, we do not know yet whether $(\text{ISO}(\mathbb{Z}^n), \circ)$ is a group or whether $\text{ISO}(\mathbb{Z}^n) \subseteq \text{F}(\mathbb{Z}^n)$. It turns out that the results are indeed positive, i.e. $\text{ISO}(\mathbb{Z}^n) \leq \text{F}(\mathbb{Z}^n)$, but we need more steps to see this.

Definition 9.5. Let $G \leq \text{F}(X)$, $Y \subseteq X$, and $G_Y := \{g \in G \mid g(Y) = Y\}$ be the setwise stabilizer of Y under G . The restricted setwise stabilizer of Y under G is the set

$$G_Y|_Y := \{g|_Y \mid g \in G_Y\}, \quad (9.31)$$

where $g|_Y : Y \rightarrow Y$ is the (surjective) restriction of the function g to Y .

Theorem 9.6. For any $Y \subseteq X$, $\text{F}(Y) = \text{F}(X)_Y|_Y$.

Proof. Pick any $f' \in \text{F}(Y)$, and let $f : X \rightarrow X$ be the function given by

$$f(x) = \begin{cases} f'(x), & x \in Y; \\ x, & x \in X \setminus Y. \end{cases} \quad (9.32)$$

Then it is clear that $f(Y) = f'(Y) = Y$ and $f(X \setminus Y) = X \setminus Y$. For any $x, x' \in X$ and $f(x) = f(x')$: if $f(x) \in X \setminus Y$ then $x = x'$; otherwise $x, x' \in Y$ and $f'(x) = f'(x')$ which yields $x = x'$ since f' is injective. This implies that f is injective. f is also surjective, since $f(X) = f(Y \cup (X \setminus Y)) = f(Y) \cup f(X \setminus Y) = Y \cup (X \setminus Y) = X$. So $f \in \text{F}(X)$. Further, the fact that $f(Y) = f'(Y) = Y$ implies that $f \in \text{F}(X)_Y$ and $f' = f|_Y \in \text{F}(X)_Y|_Y$. Therefore, $\text{F}(Y) \subseteq \text{F}(X)_Y|_Y$. Conversely, pick any $f|_Y \in \text{F}(X)_Y|_Y$. $f|_Y$ is injective since $f \in \text{F}(X)_Y \subseteq \text{F}(X)$ is injective; $f|_Y$ is surjective since $f|_Y(Y) = f(Y) = Y$. So $f|_Y \in \text{F}(Y)$. This implies that $\text{F}(X)_Y|_Y \subseteq \text{F}(Y)$.

Corollary 9.1. $F(\mathbb{Z}^n) = F(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$.

Theorem 9.7. $T(\mathbb{Z}^n) = T(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$, and $R(\mathbb{Z}^n) = R(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$.

Proof. Pick any $t'_u \in T(\mathbb{Z}^n)$, then by definition, $u \in \mathbb{Z}^n$, and $t'_u(x) = x + u$, for any $x \in \mathbb{Z}^n$. Let $t : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be the function given by $t(x) = x + u$. Since $u \in \mathbb{Z}^n$, then $u \in \mathbb{R}^n$, so $t \in T(\mathbb{R}^n)$. Further, note that $t(\mathbb{Z}^n) = \mathbb{Z}^n$; therefore, $t \in T(\mathbb{R}^n)_{\mathbb{Z}^n}$. It follows that $t'_u = t|_{\mathbb{Z}^n} \in T(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$, so $T(\mathbb{Z}^n) \subseteq T(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$.

Pick any $t' \in T(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$, then by definition, there exists a $t_u \in T(\mathbb{R}^n)$ where $u \in \mathbb{R}^n$ such that $t_u(\mathbb{Z}^n) = \mathbb{Z}^n$ and $t' = t_u|_{\mathbb{Z}^n}$, i.e. $t'(x) = x + u$, for any $x \in \mathbb{Z}^n$. The condition $t_u(\mathbb{Z}^n) = \mathbb{Z}^n$ implies in particular $t_u(\mathbf{0}) = u \in \mathbb{Z}^n$. It follows that $t' \in T(\mathbb{Z}^n)$, so $T(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n} \subseteq T(\mathbb{Z}^n)$.

Pick any $r'_A \in R(\mathbb{Z}^n)$, then by definition, $A \in O_n(\mathbb{Z})$, and $r'_A(x) = Ax$, for any $x \in \mathbb{Z}^n$. Let $r : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be the function given by $r(x) = Ax$. Since $A \in O_n(\mathbb{Z})$, then $A \in O_n(\mathbb{R})$, so $r \in R(\mathbb{R}^n)$. Further, note that $r(\mathbb{Z}^n) = \mathbb{Z}^n$; therefore, $r \in R(\mathbb{R}^n)_{\mathbb{Z}^n}$. It follows that $r'_A = r|_{\mathbb{Z}^n} \in R(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$, so $R(\mathbb{Z}^n) \subseteq R(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$.

Pick any $r' \in R(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$, then by definition, there exists a $r_A \in R(\mathbb{R}^n)$ where $A \in O_n(\mathbb{R})$ such that $r_A(\mathbb{Z}^n) = \mathbb{Z}^n$ and $r' = r_A|_{\mathbb{Z}^n}$, i.e. $r'(x) = Ax$, for any $x \in \mathbb{Z}^n$. The condition $r_A(\mathbb{Z}^n) = \mathbb{Z}^n$ implies in particular $A\mathbf{e}_i \in \mathbb{Z}^n$ for all i , i.e. the columns of A are from \mathbb{Z}^n . So $A \in O_n(\mathbb{Z})$. It follows that $r' \in R(\mathbb{Z}^n)$, so $R(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n} \subseteq R(\mathbb{Z}^n)$.

Theorem 9.8. $ISO(\mathbb{Z}^n) = ISO(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$.

Proof. Pick any $h' \in ISO(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n}$, then by definition, there exists an $h \in ISO(\mathbb{R}^n)$ such that $h(\mathbb{Z}^n) = \mathbb{Z}^n$ and $h' = h|_{\mathbb{Z}^n}$. For any $x, y \in \mathbb{Z}^n$,

$$d(h'(x), h'(y)) = d(h|_{\mathbb{Z}^n}(x), h|_{\mathbb{Z}^n}(y)) = d(h(x), h(y)) = d(x, y). \quad (9.33)$$

This implies that $h' \in ISO(\mathbb{Z}^n)$. So, $ISO(\mathbb{R}^n)_{\mathbb{Z}^n}|_{\mathbb{Z}^n} \subseteq ISO(\mathbb{Z}^n)$.

Conversely, pick any $h' \in ISO(\mathbb{Z}^n)$ and let $h'_0 = h' - h'(\mathbf{0})$. Note that $h'_0 \in ISO(\mathbb{Z}^n)$ and $h'_0(\mathbf{0}) = \mathbf{0}$. This implies that $\|h'_0(x)\|_2 = d(h'_0(x), h'_0(\mathbf{0})) = d(x, \mathbf{0}) = \|x\|_2$, for any $x \in \mathbb{Z}^n$. Further, for any $x, y \in \mathbb{Z}^n$, expanding the distance-preserving equation $\|h'_0(x) - h'_0(y)\|_2^2 = \|x - y\|_2^2$ and cancelling equal terms (i.e. $\|h'_0(x)\|_2^2 = \|x\|_2^2$ and $\|h'_0(y)\|_2^2 = \|y\|_2^2$) yields

$$\langle h'_0(x), h'_0(y) \rangle = \langle x, y \rangle \quad \text{for all } x, y \in \mathbb{Z}^n. \quad (9.34)$$

Now let $h : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be the function given by $h(x) = Ax + u$, where $A = [h'_0(\mathbf{e}_1), \dots, h'_0(\mathbf{e}_n)] \in \mathbb{Z}^{n \times n}$ and $u = h'(\mathbf{0}) \in \mathbb{Z}^n$. Moreover, $\langle h'_0(\mathbf{e}_i), h'_0(\mathbf{e}_j) \rangle = \langle \mathbf{e}_i, \mathbf{e}_j \rangle = \delta_{ij}$. So, A is orthogonal, i.e. $A \in O_n(\mathbb{Z}) \subseteq O_n(\mathbb{R})$. This implies $h \in ISO(\mathbb{R}^n)$. We claim $h(\mathbb{Z}^n) = \mathbb{Z}^n$ and $h' = h|_{\mathbb{Z}^n}$.

To see $h(\mathbb{Z}^n) = \mathbb{Z}^n$, first pick any $x \in h(\mathbb{Z}^n)$, then there exists $y \in \mathbb{Z}^n$ such that $x = h(y) = Ay + u$. Since $A \in \mathbb{Z}^{n \times n}$ and $u \in \mathbb{Z}^n$, $x \in \mathbb{Z}^n$ which implies $h(\mathbb{Z}^n) \subseteq \mathbb{Z}^n$. Conversely, pick any $x \in \mathbb{Z}^n$. Let $y = A^\top(x - u)$, then $y \in \mathbb{Z}^n$ and $h(y) = Ay + u = x$. So $x \in h(\mathbb{Z}^n)$ which implies $\mathbb{Z}^n \subseteq h(\mathbb{Z}^n)$.

To see $h' = h|_{\mathbb{Z}^n}$, pick any $x \in \mathbb{Z}^n$, then $\langle h'_0(\mathbf{e}_i), h'_0(x) \rangle = \langle \mathbf{e}_i, x \rangle = \langle A\mathbf{e}_i, Ax \rangle = \langle h'_0(\mathbf{e}_i), Ax \rangle$, for all $i = 1, \dots, n$. So, $\langle h'_0(\mathbf{e}_i), h'_0(x) - Ax \rangle = 0$, for all $i = 1, \dots, n$, i.e.

$$A^\top(h'_0(x) - Ax) = \mathbf{0}. \quad (9.35)$$

Multiplying both sides by A yields $h'_0(x) = Ax$, for all $x \in \mathbb{Z}^n$. Hence,

$$h(x) = Ax + u = h'_0(x) + h'(\mathbf{0}) = h'(x) \quad \text{for all } x \in \mathbb{Z}^n. \quad (9.36)$$

That is, $h' = h|_{\mathbb{Z}^n}$. It follows that $h' \in \text{ISO}(\mathbb{R}^n)_{\mathbb{Z}^n|_{\mathbb{Z}^n}}$. So, $\text{ISO}(\mathbb{Z}^n) \subseteq \text{ISO}(\mathbb{R}^n)_{\mathbb{Z}^n|_{\mathbb{Z}^n}}$.

Remark 9.5. *Through restricted setwise stabilizers, Corollary 9.1 as well as Theorems 9.7 and 9.8 collectively verify that transformations, translations, rotations, and isometries of \mathbb{Z}^n are precisely those transformations, translations, rotations, and isometries of \mathbb{R}^n that stabilize \mathbb{Z}^n , respectively. In particular, it is now clear that $(\text{ISO}(\mathbb{Z}^n), \circ)$ is indeed a group, and moreover $\text{T}(\mathbb{Z}^n), \text{R}(\mathbb{Z}^n) \leq \text{ISO}(\mathbb{Z}^n) \leq \text{F}(\mathbb{Z}^n)$.*

The parallels between translations, rotations, isometries of \mathbb{Z}^n and their counterparts of \mathbb{R}^n yield the two characterizations of $\text{ISO}(\mathbb{Z}^n)$ which are parallel to the those of $\text{ISO}(\mathbb{R}^n)$:

$$\text{ISO}(\mathbb{Z}^n) = \langle \text{T}(\mathbb{Z}^n) \cup \text{R}(\mathbb{Z}^n) \rangle \quad \text{where } \text{T}(\mathbb{Z}^n) \cap \text{R}(\mathbb{Z}^n) = \{\text{id}\}; \quad (9.37)$$

$$\text{ISO}(\mathbb{Z}^n) = \text{T}(\mathbb{Z}^n) \circ \text{R}(\mathbb{Z}^n) \cong \mathbb{Z}^n \rtimes \text{O}_n(\mathbb{Z}). \quad (9.38)$$

This further yields the parametrization of $\mathcal{H}_{\text{ISO}(\mathbb{Z}^n)}^*$ by

$$\Sigma''_{\text{isozn}} := \{(L, V, \xi) \mid L \leq \text{O}_n(\mathbb{Z}), V \leq \mathbb{Z}^n, \xi \in \Xi''_{L,V}\} \subseteq \Sigma' \subseteq \Sigma, \quad (9.39)$$

where $\Xi''_{L,V} := \{\xi \in \Xi_{L,V} \mid \xi(L) \subseteq \mathbb{Z}^n/V\} \subseteq \Xi_{L,V}$. Note that $\text{ISO}(\mathbb{Z}^n)$ still have infinitely many subgroups, since the choices for V and ξ are still unlimited. Next we will show how to enumerate a finite subset from $\mathcal{H}_{\text{ISO}(\mathbb{Z}^n)}^*$ when considering the music domain.

The space of music pitches from equal temperament can be denoted by \mathbb{Z} . Every adjacent pitch is separated by a half-step (or semi-tone) denoted by the integer 1, which is also the distance between every adjacent keys (regardless of black or white) in a piano keyboard. While the absolute integer assigned to each music pitch is not essential, in the standard

MIDI convention, C4 is 60, C♯4 is 61, and so forth. Therefore, the space \mathbb{Z}^n represents the space of chords consisting of n pitches. For instance, \mathbb{Z}^3 denotes the space of trichords, \mathbb{Z}^4 denotes the space of tetrachords, and so forth. Known music transformations of fixed-size chords [84, 85] can be summarized as a subset of the following parametrization set

$$\Sigma''_{music} := \{(L, V, \xi) \mid L \leq \mathbf{O}_n(\mathbb{Z}), V \in \mathcal{H}_{\mathbb{Z}^n}^M, \xi = \xi_{L,V}^0 \in \Xi''_{L,V}\} \subseteq \Sigma''_{isozn}, \quad (9.40)$$

where $\mathcal{H}_{\mathbb{Z}^n}^M := \{\langle \mathbf{1} \rangle, (12\mathbb{Z})^n, \langle \mathbf{1} \rangle \vee (12\mathbb{Z})^n\}$ is a finite collection of music translation subgroups including music transpositions, octave shifts, and their combinations; $\xi_{L,V}^0$ is the trivial vector system given by $\xi_{L,V}^0(A) = V$ for any $A \in L$ requiring the inclusion of all rotations to include music permutations and inversions. Together with the fact that $\mathbf{O}_n(\mathbb{Z})$ is finite, the enumeration of each element in the triplet (L, V, ξ) is finite, yielding a finite Σ''_{music} .

It is important to recognize that the significance of the parametrization set Σ''_{music} is not limited to recover known music-theoretic concepts but to complete existing knowledge by forming a music “closure” Σ''_{music} . Such a “closure” can be further fine-tuned to be either more efficient (e.g. by removing uninteresting rotation subgroups) or more expressive (e.g. by adding more translation subgroups).

9.2.4 Section Summary

In this section, we first moved down from the full transformation group of \mathbb{R}^n —the top vertex in the subgroup lattice $(\mathcal{H}_{\mathbb{F}(\mathbb{R}^n)}^*, \leq)$ —to the affine group of \mathbb{R}^n . Focusing on $\mathbf{AFF}(\mathbb{R}^n)$, we derived a complete identification of its subgroups by constructing a parametrization set Σ and a bijection $\Psi : \mathcal{H}_{\mathbf{AFF}(\mathbb{R}^n)}^* \rightarrow \Sigma$. So, every subgroup of $\mathbf{AFF}(\mathbb{R}^n)$ bijectively corresponds to a unique triplet in Σ . Towards the goal of a finite collection of affine subgroups, we further moved down in the subgroup lattice $(\mathcal{H}_{\mathbb{F}(\mathbb{R}^n)}^*, \leq)$ from the affine group of \mathbb{R}^n to the isometry group of \mathbb{R}^n . Focusing on $\mathbf{ISO}(\mathbb{R}^n)$, we identified the parametrization of $\mathcal{H}_{\mathbf{ISO}(\mathbb{R}^n)}^*$ by a subset $\Sigma' \subseteq \Sigma$. From there, we made a dichotomy in our top-down path, and presented two examples to obtain two collections of subgroups used in two different topic domains. One is a finite collection of space groups (in standard form and up to affine conjugacy) used in crystallography, which is parameterized by $\Sigma''_{crist} \subseteq \Sigma'$; the other is a finite completion of existing music concepts, which is parameterized by $\Sigma''_{music} \subseteq \Sigma''_{isozn} \subseteq \Sigma'$. A complete roadmap that we have gone through is summarized in Figure 9.3.

We finally reiterate that the selection of top-down paths is one’s design choice. Whenever necessary, one should make his/her own decision on creating a new branch or even trying out several branches along major downward paths. The top-down path with two branches

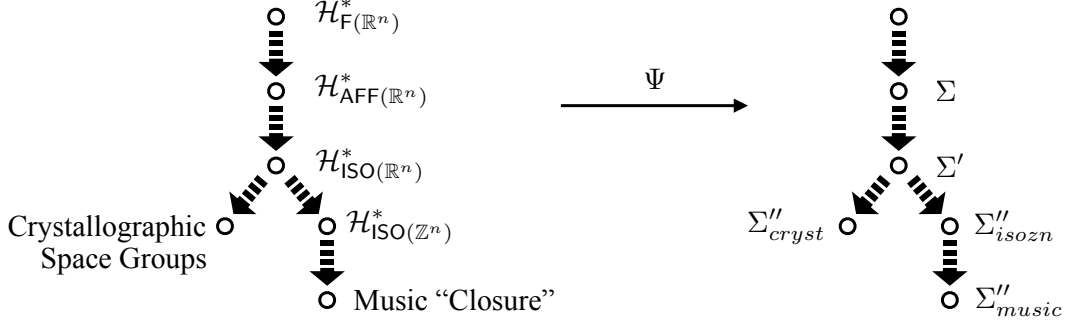


Figure 9.3: Roadmap of the top-down paths in terms of collection of subgroups (left) and the corresponding parametrization paths (right) under the parametrization map Ψ .

introduced in this section serve for illustration purposes.

9.3 SYMMETRY GENERATION: BOTTOM-UP APPROACH

As the counterpart of Section 9.2, this section presents the bottom-up approach to sample a finite subgroup sublattice from the complete lattice $\mathcal{H}_{F(X)}^*$. More specifically, we extract a partial subgroup lattice $\mathcal{H}_{\langle S \rangle}$ from a generating set S . This is done by an induction procedure which first extracts cyclic subgroups $\{\langle s \rangle \mid s \in S\}$ as base cases, and then inductively extracts other subgroups via the join of the extracted ones. The resulting collection of subgroups $\mathcal{H}_{\langle S \rangle}$ is generally not the complete subgroup lattice $\mathcal{H}_{\langle S \rangle}^*$ since some of its subgroups are missing. The dual of this induction procedure gives a mirrored induction algorithm that computes the corresponding abstraction semiuniverse in an efficient way. The missing subgroups can be made up by adding more generators, but this hinders the efficiency. At the end of this section, we will discuss the trade-off between expressiveness and efficiency when designing a generating set in practice.

9.3.1 From Generating Set to Subgroup (Semi)Lattice

Let $S \subseteq F(X)$ be a finite subset consisting of transformations of a set X . We construct a collection $\mathcal{H}_{\langle S \rangle}$ consisting of subgroups of $\langle S \rangle$ where every subgroup is generated by a subset of S . To succinctly record this process and concatenate it with the abstraction generating chain, we introduce the following one-step *subgroup generating chain*:

$$\text{a subset of } F(X) \xrightarrow{\text{group generation}} \text{a subgroup of } F(X), \quad (9.41)$$

which can be further encapsulated by the *subgroup generating function* defined as follows.

Definition 9.6. *The subgroup generating function is the mapping $\pi' : 2^{\mathbf{F}(X)} \rightarrow \mathcal{H}_{\mathbf{F}(X)}^*$ where $2^{\mathbf{F}(X)}$ is the power set of $\mathbf{F}(X)$, $\mathcal{H}_{\mathbf{F}(X)}^*$ is the collection of all subgroups of $\mathbf{F}(X)$, and for any $S \in 2^{\mathbf{F}(X)}$, $\pi'(S) := \langle S \rangle := \{s_k \circ \cdots \circ s_1 \mid s_i \in S \cup S^{-1}, i = 1, \dots, k, k \in \mathbb{Z}_{\geq 0}\}$ where $S^{-1} := \{s^{-1} \mid s \in S\}$. By convention, $s_k \circ \cdots \circ s_1 = \text{id}$ for $k = 0$, and $\pi'(\emptyset) = \langle \emptyset \rangle = \{\text{id}\}$.*

Remark 9.6. *The subgroup generating function in Definition 9.6 is nothing but generating a subgroup from its given generating set. However, we can now write the procedure at the beginning of this subsection succinctly as $\mathcal{H}_{\langle S \rangle} = \pi'(2^S)$ for any finite subset $S \subseteq \mathbf{F}(X)$; further, the subgroup generating chain and the abstraction generating chain can now be concatenated, which is denoted by the composition $\Pi := \pi \circ \pi'$.*

Like the abstraction generating function π , the subgroup generating function π' is not necessarily injective, since a generating set of a group is generally not unique; π' is surjective, since every subgroup per se is also its own generating set. The following theorem captures the structure of $\pi'(2^S)$ for a finite subset $S \subseteq \mathbf{F}(X)$.

Theorem 9.9. *Let $S \subseteq \mathbf{F}(X)$ be a finite subset, and π' be the subgroup generating function. Then $(\pi'(2^S), \leq)$ is a join-semilattice, but not necessarily a meet-semilattice. In particular,*

$$\pi'(A \cup B) = \pi'(A) \vee \pi'(B) \quad \text{for any } A, B \subseteq S. \quad (9.42)$$

Proof. For any $A, B \subseteq S$, we have

$$\pi'(A \cup B) = \langle A \cup B \rangle = \langle \langle A \rangle \cup \langle B \rangle \rangle = \langle \pi'(A) \cup \pi'(B) \rangle = \pi'(A) \vee \pi'(B). \quad (9.43)$$

Then for any $\pi'(A), \pi'(B) \in \pi'(2^S)$ where $A, B \subseteq S$, the join $\pi'(A) \vee \pi'(B) = \pi'(A \cup B) \in \pi'(2^S)$, since $A \cup B \subseteq S$. So, $(\pi'(2^S), \leq)$ is a join-semilattice.

We give an example in which $(\pi'(2^S), \leq)$ is not a meet-semilattice. Let $X = \mathbb{R}^n$ and $S = \{t_{\mathbf{e}_1}, t_{\mathbf{e}_2}, t_{(3/2)\cdot \mathbf{1}}\}$ be a set consisting of three translations where $\mathbf{e}_1 = (1, 0)$, $\mathbf{e}_2 = (0, 1)$, $\mathbf{1} = (1, 1)$. Further, let $A = \{t_{\mathbf{e}_1}, t_{\mathbf{e}_2}\}$ and $B = \{t_{(3/2)\cdot \mathbf{1}}\}$. The meet $\pi'(A) \wedge \pi'(B) = \langle A \rangle \cap \langle B \rangle = \langle t_{\mathbf{3}\cdot \mathbf{1}} \rangle \notin \pi'(2^S)$.

Remark 9.7. *Although the collection of subgroups generated by the subgroup generating function π' is not a lattice in general, it is sufficient that it is a join-semilattice. This is because the family of abstractions generated by the abstraction generating function π is a meet-semiuniverse (recall the strong and weak dualities in Theorem 6.3). As a result, the closedness of $\pi'(2^S)$ under join is carried over through the strong duality to preserve*

the closedness of $\pi(\pi'(2^S))$ under meet. This preservation of closednesses under join and meet has a significant practical implication: it directly yields an induction algorithm that implements $\Pi(2^S) := \pi \circ \pi'(2^S)$ from a finite subset S .

9.3.2 An Induction Algorithm

We describe an algorithmic implementation of $\Pi(2^S) := \pi \circ \pi'(2^S)$, where $S \subseteq F(X)$ as the input is a finite subset, and $\Pi(2^S)$ as the output is the resulting abstraction semiuniverse. Here we assume X is a finite space for computational feasibility. A naive implementation will first compute the subgroup join-semilattice $\pi'(2^S)$ as an intermediate step, and then compute the abstraction meet-semiuniverse $\pi(\pi'(2^S))$ as the second step. However, as mentioned in the remark following Theorem 6.3, computing every abstraction of X by identifying orbits from a subgroup action can be expensive, and even computationally prohibitive. In this subsection, we first analyze the naive two-step implementation, and then introduce an induction algorithm that efficiently computes $\Pi(2^S)$ without the intermediate step, avoiding expensive computations for all abstractions from orbits identifications.

A Naive Two-Step Implementation. For a given input $S \subseteq F(X)$ where both S and X are finite, we first compute $\pi'(2^S)$ which is straightforward, since we can simply index (possibly with duplication) every subgroup in $\pi'(2^S)$ by its generating set $S' \subseteq S$. Now consider the second step: for every $S' \subseteq S$ and its corresponding $\pi'(S') = \langle S' \rangle$, we compute $\pi(\pi'(S')) = X/\langle S' \rangle$ by identifying the set of orbits $\{\langle S' \rangle x \mid x \in X\}$. More specifically, for every pair $x, x' \in X$, we need to check whether or not they are in the same orbit. The number of checks needed is $O(|X|^2)$ which can be computationally prohibitive if $|X|$ is large. Nevertheless, what really makes this naive thought fail is that most checks cannot finish in finite time. Take $S' = \{s_1, s_2\}$ for example, without additional properties to leverage, there are infinitely many ways of causing x, x' to be in the same orbit, e.g. $x' = s_1(x)$, $x' = s_1^{-1}(x)$, $x' = s_2(x)$, $x' = s_1 \circ s_2 \circ s_2 \circ s_1^{-1}(x)$ and so forth.

An Induction Algorithm. Instead, we give an algorithm based on induction on $|S'|$ for all nonempty subsets $S' \subseteq S$.

Base case: compute $\Pi(S')$ for $|S'| = 1$ (Algorithm 9.1) as orbits under a cyclic subgroup:

$$\Pi(S') = \{\langle S' \rangle x \mid x \in X\}. \quad (9.44)$$

Induction step: compute $\Pi(S')$ for $|S'| > 1$ (Algorithm 9.2) as the meet of two partitions:

$$\Pi(S') = \Pi(S'') \wedge \Pi(S' \setminus S'') \quad \text{for any } S'' \subset S'. \quad (9.45)$$

In the base case, every base partition is generated through orbits identification (or more precisely, orbits tracing), which however does not require any endless checks since there is only one generator. As a result, the computational complexity of Algorithm 9.1 is linear rather than quadratic in the size of the set X . The correctness of the induction step is backed by Theorem 9.9 and the strong duality in Theorem 6.3, or more explicitly,

$$\Pi(S') = \pi \circ \pi'(S'' \cup (S' \setminus S'')) \quad (9.46)$$

$$= \pi(\pi'(S'') \vee \pi'(S' \setminus S'')) \quad (9.47)$$

$$= \pi \circ \pi'(S'') \wedge \pi \circ \pi'(S' \setminus S'') \quad (9.48)$$

$$= \Pi(S'') \wedge \Pi(S' \setminus S''). \quad (9.49)$$

It is the meet operation that successfully bypasses the endless checks in the naive implementation.

Input: a generator s and a set X
Output: the base partition $\Pi(\{s\})$

Function BasePartn(s):

```

initialize label id:  $l = 0$ ;
for each point  $x \in X$  do
    if  $x$  is not labelled then
        initialize a new orbit:
             $O = \{x\}$ ;
        transform:
             $y = s(x)$ ;
        while  $y \in X$  and  $y \neq x$ 
            and  $y$  is not labelled do
                enlarge the orbit:
                     $O = O \cup \{y\}$ ;
                transform:
                     $y = s(y)$ ;
            end
        if  $y \notin X$  or  $y = x$  then
            create a new label:
                 $l = l + 1$ ;
        end
        if  $y$  is labelled then
            use  $y$ 's label:
                 $l = y$ 's label;
        end
        label every point in the
        orbit  $O$  by  $l$ ;
    end
end
return the partition according
to the labels;

```

Algorithm 9.1: Computing base partitions by identifying orbits: $O(|X|)$.

Input: two partitions \mathcal{P} and \mathcal{Q} of a set X

Output: the meet $\mathcal{P} \wedge \mathcal{Q}$, i.e. finest common coarsening of \mathcal{P} and \mathcal{Q}

Function Meet(\mathcal{P}, \mathcal{Q}):

```

for each cell  $Q \in \mathcal{Q}$  do
    initialize a new cell:
         $P_{merge} = \emptyset$ ;
    for each cell  $P \in \mathcal{P}$  do
        if  $P \cap Q \neq \emptyset$  then
            merge:
                 $P_{merge} = P_{merge} \cup P$ ;
            remove:
                 $\mathcal{P} = \mathcal{P} \setminus \{P\}$ ;
        end
    end
    insert:
         $\mathcal{P} = \mathcal{P} \cup \{P_{merge}\}$ ;
end
return  $\mathcal{P}$ ;

```

Algorithm 9.2: Computing partitions generated from more than one generators inductively by taking the meet of two partitions computed earlier: $O(|\mathcal{P}||\mathcal{Q}|)$. Normally, all base partitions should be already computed and cached before running the induction steps.

9.3.3 Finding a Generating Set of $\text{ISO}(\mathbb{Z}^n)$

We give an example of finding a finite generating set. The key idea is based on recursive group decompositions. In light of storing abstractions of a set X in digital computers, we consider the discrete space $X = \mathbb{Z}^n (\subseteq \mathbb{R}^n)$. Further, we restrict our attention to generators that are isometries of \mathbb{Z}^n , since $\text{ISO}(\mathbb{Z}^n)$ is finitely generated. We show this by explicitly finding a finite generating set of $\text{ISO}(\mathbb{Z}^n)$.

Recall that (in Section 9.2.3, the second example) we gave one characterization of $\text{ISO}(\mathbb{Z}^n)$:

$$\text{ISO}(\mathbb{Z}^n) = \langle \text{T}(\mathbb{Z}^n) \cup \text{R}(\mathbb{Z}^n) \rangle \quad \text{where } \text{T}(\mathbb{Z}^n) \cap \text{R}(\mathbb{Z}^n) = \{\text{id}\}. \quad (9.50)$$

We start from (9.50), and further seek a generating set of $\text{T}(\mathbb{Z}^n)$ and a generating set of $\text{R}(\mathbb{Z}^n)$. Finding generators of $\text{T}(\mathbb{Z}^n)$ is easy: $\text{T}(\mathbb{Z}^n) = \langle t'_{e_1} \cup \dots \cup t'_{e_n} \rangle$. However, finding generators of $\text{R}(\mathbb{Z}^n)$ requires more structural inspections. The strategy is to first study the matrix group $\text{O}_n(\mathbb{Z})$ which is isomorphic to $\text{R}(\mathbb{Z}^n)$, and then transfer results to $\text{R}(\mathbb{Z}^n)$. Interestingly, $\text{O}_n(\mathbb{Z})$ has a decomposition similar to what $\text{ISO}(\mathbb{Z}^n)$ has in (9.50). By definition, $\text{O}_n(\mathbb{Z})$ consists of all orthogonal matrices with integer entries. For any $A \in \text{O}_n(\mathbb{Z})$, the orthogonality and integer-entry constraints restrict every column vector of A to be a unique standard basis vector or its negation. This will lead to the decomposition of $\text{R}(\mathbb{Z}^n)$.

Notations. $\mathbf{1} = (1, \dots, 1) \in \mathbb{R}^n$ is the all-ones vector; e_1, \dots, e_n are the standard basis vectors of \mathbb{R}^n where $e_i \in \{0, 1\}^n$ has a 1 in the i th coordinate and 0s elsewhere; ν_1, \dots, ν_n are the so-called unit negation vectors of \mathbb{R}^n where $\nu_i \in \{-1, 1\}^n$ has a -1 in the i th coordinate and 1s elsewhere.

Definition 9.7 (Permutation). *A permutation matrix is a matrix obtained by permuting the rows of an identity matrix; we denote the set of all $n \times n$ permutation matrices by P_n . A permutation of an index set is a bijection $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$; the set of all permutations of the size- n index set is known as the symmetric group S_n . A permutation of (integer-valued) vectors is a rotation $r'_P : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ for some $P \in \text{P}_n$; we denote the set of all permutations of n -dimensional vectors by $\text{R}_P(\mathbb{Z}^n) \subseteq \text{R}(\mathbb{Z}^n)$.*

Definition 9.8 (Negation). *A (partial) negation matrix is a diagonal matrix whose diagonal entries are drawn from $\{-1, 1\}$; we denote the set of all $n \times n$ negation matrices by N_n . A (partial) negation of (integer-valued) vectors is a rotation $r'_N : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ for some $N \in \text{N}_n$; we denote the set of all negations of n -dimensional vectors by $\text{R}_N(\mathbb{Z}^n) \subseteq \text{R}(\mathbb{Z}^n)$.*

Remark 9.8. *Under Definitions 9.7 and 9.8, one can verify that a permutation (of vectors) maps x to Px by permuting x 's coordinates according to $P \in \text{P}_n$; likewise, a negation (of vectors) maps x to Nx by negating x 's coordinates according to $N \in \text{N}_n$.*

Theorem 9.10. *We have the following characterizations of permutations and negations:*

$$(\mathbf{R}_P(\mathbb{Z}^n), \circ) \cong (\mathbf{P}_n, \cdot) \cong (S_n, \circ) \quad \text{and} \quad (\mathbf{R}_N(\mathbb{Z}^n), \circ) \cong (\mathbf{N}_n, \cdot). \quad (9.51)$$

In particular, these imply that $|\mathbf{R}_P(\mathbb{Z}^n)| = |\mathbf{P}_n| = |S_n| = n!$ and $|\mathbf{R}_N(\mathbb{Z}^n)| = |\mathbf{N}_n| = 2^n$.

Proof. It is an exercise to check that all entities in the theorem are indeed groups.

Let $\phi_P : \mathbf{R}_P(\mathbb{Z}^n) \rightarrow \mathbf{P}_n$ be the function given by $\phi_P(r'_P) = P$, for any $r'_P \in \mathbf{R}_P(\mathbb{Z}^n)$. For any $r'_P, r'_Q \in \mathbf{R}_P(\mathbb{Z}^n)$, if $\phi_P(r'_P) = \phi_P(r'_Q)$, i.e. $P = Q$, then $r'_P = r'_Q$, so ϕ_P is injective. For any $P \in \mathbf{P}_n$, $r'_P \in \mathbf{R}_P(\mathbb{Z}^n)$ and $\phi_P(r'_P) = P$, so ϕ_P is surjective. Further, for any $r'_P, r'_Q \in \mathbf{R}_P(\mathbb{Z}^n)$, $\phi_P(r'_P \circ r'_Q) = \phi_P(r'_{P \cdot Q}) = P \cdot Q = \phi_P(r'_P) \cdot \phi_P(r'_Q)$, so ϕ_P is a homomorphism. Now we see that ϕ_P is an isomorphism. So, $(\mathbf{R}_P(\mathbb{Z}^n), \circ) \cong (\mathbf{P}_n, \cdot)$.

Let $\phi_S : S_n \rightarrow \mathbf{P}_n$ be the function given by $\sigma \mapsto P^\sigma$, where P^σ is an $n \times n$ permutation matrix obtained by permuting the rows of the identity matrix according to σ , i.e.

$$P_{ij}^\sigma = \begin{cases} 1 & i = \sigma(j) \\ 0 & i \neq \sigma(j) \end{cases} \quad \text{for any } i, j \in \{1, \dots, n\}. \quad (9.52)$$

For any $\sigma, \mu \in S_n$, if $\phi_S(\sigma) = \phi_S(\mu)$, i.e. $P^\sigma = P^\mu$, then $\sigma(j) = \mu(j)$ for all $j \in \{1, \dots, n\}$, i.e. $\sigma = \mu$, so ϕ_S is injective. For any $P \in \mathbf{P}_n$, let $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ be the function given by $\sigma(j) \in \{i | P_{ij} = 1\}$, which is well-defined since $\{i | P_{ij} = 1\}$ is a singleton for all $j \in \{1, \dots, n\}$ given that $P \in \mathbf{P}_n$ is a permutation matrix. It is clear that $\sigma \in S_n$, and $\phi_S(\sigma) = P$. So, ϕ_S is surjective. Further, for any $\sigma, \mu \in S_n$, $\phi_S(\sigma \circ \mu) = P^{\sigma \circ \mu} = P^\sigma \cdot P^\mu = \phi_S(\sigma) \cdot \phi_S(\mu)$ where the second equality holds because for all $i, j \in \{1, \dots, n\}$,

$$(P^\sigma \cdot P^\mu)_{ij} = \sum_{k=1}^n P_{ik}^\sigma \cdot P_{kj}^\mu = P_{i\mu(j)}^\sigma \cdot 1 = \begin{cases} 1 & i = \sigma \circ \mu(j) \\ 0 & i \neq \sigma \circ \mu(j) \end{cases} = P_{ij}^{\sigma \circ \mu}, \quad (9.53)$$

so ϕ_S is a homomorphism. Now we see that ϕ_S is an isomorphism. So, $(S_n, \circ) \cong (\mathbf{P}_n, \cdot)$.

Let $\phi_N : \mathbf{R}_N(\mathbb{Z}^n) \rightarrow \mathbf{N}_n$ be the function given by $\phi_N(r'_N) = N$, for any $r'_N \in \mathbf{R}_N(\mathbb{Z}^n)$. For any $r'_N, r'_M \in \mathbf{R}_N(\mathbb{Z}^n)$, if $\phi_N(r'_N) = \phi_N(r'_M)$, i.e. $N = M$, then $r'_N = r'_M$, so ϕ_N is injective. For any $N \in \mathbf{N}_n$, $r'_N \in \mathbf{R}_N(\mathbb{Z}^n)$ and $\phi_N(r'_N) = N$, so ϕ_N is surjective. Further, for any $r'_N, r'_M \in \mathbf{R}_N(\mathbb{Z}^n)$, $\phi_N(r'_N \circ r'_M) = \phi_N(r'_{N \cdot M}) = N \cdot M = \phi_N(r'_N) \cdot \phi_N(r'_M)$, so ϕ_N is a homomorphism. Now we see that ϕ_N is an isomorphism. So, $(\mathbf{R}_N(\mathbb{Z}^n), \circ) \cong (\mathbf{N}_n, \cdot)$.

Theorem 9.11. *We have the following characterization of $\mathbf{O}_n(\mathbb{Z})$:*

$$\mathbf{O}_n(\mathbb{Z}) = \langle \mathbf{N}_n \cup \mathbf{P}_n \rangle \quad \text{where } \mathbf{N}_n \cap \mathbf{P}_n = \{I\}. \quad (9.54)$$

Proof. We first show that $\mathbf{N}_n, \mathbf{P}_n \leq \mathbf{O}_n(\mathbb{Z})$. $(\mathbf{O}_n(\mathbb{Z}), \cdot)$ is a group since matrix multiplication \cdot is associative, $I \in \mathbf{O}_n(\mathbb{Z})$ is the identity element, and for any $A \in \mathbf{O}_n(\mathbb{Z})$, $A^\top \in \mathbf{O}_n(\mathbb{Z})$ is its inverse. Pick any $N \in \mathbf{N}_n$, then $N \in \mathbb{Z}^{n \times n}$ and $N^\top N = NN^\top = I$, so $N \in \mathbf{O}_n(\mathbb{Z})$, which implies that $\mathbf{N}_n \subseteq \mathbf{O}_n(\mathbb{Z})$. Pick any $P \in \mathbf{P}_n$, then $P = [e_{\sigma(1)}, \dots, e_{\sigma(n)}] \in \mathbb{Z}^{n \times n}$ for some $\sigma \in S_n$ and $(P^\top P)_{ij} = e_{\sigma(i)}^\top e_{\sigma(j)} = \delta_{ij}$, i.e. $P^\top P = I$, so $P \in \mathbf{O}_n(\mathbb{Z})$, which implies that $\mathbf{P}_n \subseteq \mathbf{O}_n(\mathbb{Z})$. Now we perform subgroup tests to show that $\mathbf{N}_n, \mathbf{P}_n \leq \mathbf{O}_n(\mathbb{Z})$. First, we check that 1) $I \in \mathbf{N}_n$, 2) for any $N, N' \in \mathbf{N}_n$, $NN' \in \mathbf{N}_n$, 3) for any $N \in \mathbf{N}_n$, $N^{-1} = N \in \mathbf{N}_n$; therefore, $\mathbf{N}_n \leq \mathbf{O}_n(\mathbb{Z})$. Second, we check that 1) $I \in \mathbf{P}_n$, 2) for any $P, P' \in \mathbf{P}_n$, $PP' \in \mathbf{P}_n$, 3) for any $P \in \mathbf{P}_n$, $P^{-1} = P^\top \in \mathbf{P}_n$; therefore, $\mathbf{P}_n \leq \mathbf{O}_n(\mathbb{Z})$.

Now we show that $\mathbf{N}_n \cap \mathbf{P}_n = \{I\}$. Pick any $N \in \mathbf{N}_n \setminus \{I\}$ and any $P \in \mathbf{P}_n$. It is clear that $N \neq P$ since N has at least one -1 entries while P has no -1 entries. This implies that $(\mathbf{N}_n \setminus \{I\}) \cap \mathbf{P}_n = \emptyset$. Further, $I \in \mathbf{N}_n \cap \mathbf{P}_n$. Therefore, $\mathbf{N}_n \cap \mathbf{P}_n = \{I\}$.

Lastly we show that $\mathbf{O}_n(\mathbb{Z}) = \langle \mathbf{N}_n \cup \mathbf{P}_n \rangle$. It is clear that $\langle \mathbf{N}_n \cup \mathbf{P}_n \rangle \subseteq \mathbf{O}_n(\mathbb{Z})$, since $\mathbf{N}_n, \mathbf{P}_n \leq \mathbf{O}_n(\mathbb{Z})$. Conversely, pick any $A = [\mathbf{a}_1, \dots, \mathbf{a}_n] \in \mathbf{O}_n(\mathbb{Z})$ where \mathbf{a}_i denotes the i th column of A . By definition, $A^\top A = I$, so $\langle \mathbf{a}_i, \mathbf{a}_i \rangle = \|\mathbf{a}_i\|_2^2 = 1$ for $i \in \{1, \dots, n\}$, and $\langle \mathbf{a}_i, \mathbf{a}_j \rangle = 0$ for $i, j \in \{1, \dots, n\}$ and $i \neq j$. On the one hand, given $A \in \mathbb{Z}^{n \times n}$, the unit-norm property $\|\mathbf{a}_i\|_2^2 = 1$ implies that \mathbf{a}_i is a standard basis vector or its negation, i.e. $\mathbf{a}_i = \pm \mathbf{e}_k$ for some $k \in \{1, \dots, n\}$. On the other hand, for $i \neq j$, the orthogonality property $\langle \mathbf{a}_i, \mathbf{a}_j \rangle = 0$ implies that for some $k \neq k'$, $\mathbf{a}_i = \pm \mathbf{e}_k$ and $\mathbf{a}_j = \pm \mathbf{e}_{k'}$. Thus, there exist some vector $\alpha = (\alpha_1, \dots, \alpha_n) \in \{1, -1\}^n$ and some permutation $\sigma \in S_n$ such that $A = [\alpha_1 \mathbf{e}_{\sigma(1)}, \dots, \alpha_n \mathbf{e}_{\sigma(n)}] = \text{diag}(\alpha)[e_{\sigma(1)}, \dots, e_{\sigma(n)}] = NP$, where $N = \text{diag}(\alpha) \in \mathbf{N}_n$ and $P = [e_{\sigma(1)}, \dots, e_{\sigma(n)}] \in \mathbf{P}_n$. This implies $A \in \langle \mathbf{N}_n \cup \mathbf{P}_n \rangle$. So, $\mathbf{O}_n(\mathbb{Z}) \subseteq \langle \mathbf{N}_n \cup \mathbf{P}_n \rangle$.

Corollary 9.2. *We have the following characterization of $\mathbf{R}(\mathbb{Z}^n)$:*

$$\mathbf{R}(\mathbb{Z}^n) = \langle \mathbf{R}_\mathbf{N}(\mathbb{Z}^n) \cup \mathbf{R}_\mathbf{P}(\mathbb{Z}^n) \rangle \quad \text{where } \mathbf{R}_\mathbf{N}(\mathbb{Z}^n) \cap \mathbf{R}_\mathbf{P}(\mathbb{Z}^n) = \{\text{id}\}. \quad (9.55)$$

Remark 9.9. *The decomposition of the rotation group $\mathbf{R}(\mathbb{Z}^n)$ in (9.55) has a similar form compared to the decomposition of the isometry group $\mathbf{ISO}(\mathbb{Z}^n)$ in (9.50). One can show that $\mathbf{R}(\mathbb{Z}^n)$ has a second characterization that is similar to the second characterization of $\mathbf{ISO}(\mathbb{Z}^n)$, where $\mathbf{R}(\mathbb{Z}^n)$ can also be decomposed as semi-direct products:*

$$\mathbf{R}(\mathbb{Z}^n) = \mathbf{R}_\mathbf{N}(\mathbb{Z}^n) \circ \mathbf{R}_\mathbf{P}(\mathbb{Z}^n) \cong \mathbf{N}_n \rtimes \mathbf{P}_n. \quad (9.56)$$

However, this characterization is not used in this thesis, so we omit its proof.

By Corollary 9.2, finding generators of $\mathbf{R}(\mathbb{Z}^n)$ breaks down into finding those of $\mathbf{R}_\mathbf{N}(\mathbb{Z}^n)$ and

$R_P(\mathbb{Z}^n)$, respectively. First, from unit negation vectors, we can find generators for negations:

$$R_N(\mathbb{Z}^n) = \langle \{r'_{\text{diag}(\nu_1)}, \dots, r'_{\text{diag}(\nu_n)}\} \rangle. \quad (9.57)$$

Second, from the fact that the symmetric group is generated by 2-cycles of the form $(i, i+1)$: $S_n = \langle \{(1, 2), \dots, (n-1, n)\} \rangle$ [86], we can find generators for permutations:

$$R_P(\mathbb{Z}^n) = \langle \{r'_{P(1,2)}, \dots, r'_{P(n-1,n)}\} \rangle, \quad (9.58)$$

where $P^{(i,i+1)} \in P_n$ is obtained by swapping the i th and $(i+1)$ th rows of I . Finally,

$$\begin{aligned} \text{ISO}(\mathbb{Z}^n) &= \langle T(\mathbb{Z}^n) \cup R(\mathbb{Z}^n) \rangle \\ &= \langle T(\mathbb{Z}^n) \cup R_N(\mathbb{Z}^n) \cup R_P(\mathbb{Z}^n) \rangle \\ &= \langle T_0 \cup R_{N0} \cup R_{P0} \rangle, \end{aligned} \quad (9.59)$$

where $T_0 := \{t'_{e_i}\}_{i=1}^n$, $R_{N0} := \{r'_{\text{diag}(\nu_i)}\}_{i=1}^n$, and $R_{P0} := \{r'_{P^{(i,i+1)}}\}_{i=1}^{n-1}$. Here, we performed recursive group decompositions to yield the generating set $T_0 \cup R_{N0} \cup R_{P0}$ with finite size $n + n + (n-1) = 3n - 1$. This verifies that $\text{ISO}(\mathbb{Z}^n)$ is indeed finitely generated.

9.3.4 Trade-off: Minimality or Diversity (Efficiency or Expressiveness)

A generating set of a group is not unique. There are two extremes when considering the size of a generating set. One considers the *largest* generating set of a group which is the group itself; the other considers a *minimal* generating set which is not unique either.

Definition 9.9. *Let G be a group, $S \subseteq G$, and $\langle S \rangle$ be the subgroup of G generated by S . We say that S is a minimal generating set (of $\langle S \rangle$) if for any $s \in S$, $\langle S \setminus \{s\} \rangle \neq \langle S \rangle$.*

Theorem 9.12. *Let G be a group and $S \subseteq G$, then S is a minimal generating set if and only if for any $s \in S$, $s \notin \langle S \setminus \{s\} \rangle$.*

Proof. Suppose for any $s \in S$, $s \notin \langle S \setminus \{s\} \rangle$. However, $s \in \langle S \rangle$; so, $\langle S \setminus \{s\} \rangle \neq \langle S \rangle$. By definition, S is a minimal generating set. On the other hand, suppose there exists an $s \in S$ such that $s \in \langle S \setminus \{s\} \rangle$, i.e. $s = s_k \circ \dots \circ s_1$ for some k where $s_k, \dots, s_1 \in (S \setminus \{s\}) \cup (S \setminus \{s\})^{-1}$. Pick any $s' \in \langle S \rangle$, $s' = s'_{k'} \circ \dots \circ s'_1$ for some k' where $s'_{k'}, \dots, s'_1 \in S \cup S^{-1}$. For any $i \in \{1, \dots, k'\}$, if $s'_i = s$, replace it with $s_k \circ \dots \circ s_1$; if $s'_i = s^{-1}$, replace it with $s_1^{-1} \circ \dots \circ s_k^{-1}$; otherwise $s'_i \in (S \setminus \{s\}) \cup (S \setminus \{s\})^{-1}$, do nothing. This results in an expression of s' as the composition of finitely many elements in $(S \setminus \{s\}) \cup (S \setminus \{s\})^{-1}$, i.e. $s' \in \langle S \setminus \{s\} \rangle$. So,

$\langle S \rangle \subseteq \langle S \setminus \{s\} \rangle$. It is trivial to see that $\langle S \setminus \{s\} \rangle \subseteq \langle S \rangle$ since $S \setminus \{s\} \subseteq S$. Therefore, $\langle S \setminus \{s\} \rangle = \langle S \rangle$. By definition, S is not a minimal generating set.

Considering $\text{ISO}(\mathbb{Z}^n) = \langle T_0 \cup R_{N_0} \cup R_{P_0} \rangle$, it is easy to check that T_0 , R_{N_0} , and R_{P_0} are minimal individually; whereas their union is not. Nevertheless, it is not hard to show that $S^* := \{t'_{e_1}, r'_{\text{diag}(\nu_1)}\} \cup R_{P_0}$ is a minimal generating set of $\text{ISO}(\mathbb{Z}^n)$ with size $n + 1$.

There is a trade-off between minimality and diversity, which further leads to the trade-off between efficiency and expressiveness. Again we use $\text{ISO}(\mathbb{Z}^n)$ as an example. From one extreme, a minimal generating set is most efficient in the following sense: $S \subseteq \text{ISO}(\mathbb{Z}^n)$ is a minimal generating set (of $\langle S \rangle$) if and only if $\pi'|_{2^S}$ is injective, i.e. for any $S', S'' \subseteq S$, if $S' \neq S''$, then $\pi'(S') = \langle S' \rangle \neq \langle S'' \rangle = \pi'(S'')$ (an easy check). Therefore, whenever S is not minimal, there are duplicates in the generated subgroups, and thus duplicates in the subsequent abstraction generations. Every occurrence of a duplicate is a waste of computing power since it does not produce a new abstraction in the end. Intuitively, if a generating set is further away from being minimal, then more duplicates tend to occur and the abstraction generating process is less efficient. To the other extreme, the largest generating set is most expressive in the following sense: if $S = \text{ISO}(\mathbb{Z}^n)$, then $\pi'(2^S) = \mathcal{H}_{\text{ISO}(\mathbb{Z}^n)}^*$, i.e. the collection of all subgroups of $\text{ISO}(\mathbb{Z}^n)$; and in general, for any $S \subset S_+ \subseteq \text{ISO}(\mathbb{Z}^n)$, the monotonicity property $\pi'(2^S) \subset \pi'(2^{S_+})$ holds (an easy check). However, the largest generating set is also the least efficient not only because it has the largest number of duplicates, but in this case, it is infinite. Thus, to respect the trade-off between efficiency and expressiveness, we need to find a balance between the two extremes.

Our plan is to start from a minimal generating set S^* of $\text{ISO}(\mathbb{Z}^n)$ and then gradually enlarge it by adding the so-called *derived generators*. In other words, we aim for a filtration: $S^* \subseteq S_{+1}^* \subseteq S_{+2}^* \subseteq S_{+3}^* \subseteq \dots$ such that the corresponding collections of subgroups satisfy

$$\pi'(2^{S^*}) \subseteq \pi'(2^{S_{+1}^*}) \subseteq \pi'(2^{S_{+2}^*}) \subseteq \pi'(2^{S_{+3}^*}) \subseteq \dots \quad \text{and} \quad \bigcup_{m=1}^{\infty} \pi'(2^{S_{+m}^*}) = \mathcal{H}_{\text{ISO}(\mathbb{Z}^n)}^*. \quad (9.60)$$

Definition 9.10. Let S^* be a minimal generating set of $\text{ISO}(\mathbb{Z}^n)$, and define

$$S_{+m}^* := \{s_k^{\alpha_k} \circ \dots \circ s_1^{\alpha_1} \mid k \in \mathbb{Z}_{\geq 0}, s_k, \dots, s_1 \in S^*, \alpha_k, \dots, \alpha_1 \in \mathbb{Z}, \sum_{i=1}^k |\alpha_i| \leq m\}. \quad (9.61)$$

A derived generator of length m is an $s \in S_{+m}^* \setminus S_{+(m-1)}^*$.

Remark 9.10. In Definition 9.10, S_{+m}^* is the “ball” with center id and radius m in the Cayley graph of $S^* \cup (S^*)^{-1}$. It is an easy check that $S^* \cup (S^*)^{-1} = S_{+1}^* \subseteq S_{+2}^* \subseteq S_{+3}^* \subseteq \dots$.

Note that $\cup_{m=1}^{\infty} S_{+m}^* = \langle S^* \rangle = \text{ISO}(\mathbb{Z}^n)$, since the growing “ball” will eventually cover the whole Cayley graph. Therefore, $\cup_{m=1}^{\infty} \pi'(2^{S_{+m}^*}) = \mathcal{H}_{\text{ISO}(\mathbb{Z}^n)}^*$. This suggests we gradually add derived generators of increasing length to S^* , and approximate $\mathcal{H}_{\text{ISO}(\mathbb{Z}^n)}^*$ by $\pi'(2^{S_{+m}^*})$ for some large m . Without any prior preference, one must go through this full procedure to grow the ball S_{+m}^* from radius $m = 1$. Although computationally intense, it is incremental. More importantly, this is a *one-time* procedure, but the resulting abstraction (semi)universe is *universal*: computed abstractions can be used in different topic domains.

However, just like biological perception systems which have innate preference for certain stimuli, having prior preference for certain derived generators can make the abstraction (semi)universe grow more efficiently. As an illustrative example and a design choice, we start from the minimal generating set $S^* := \{t'_{e_1}, r'_{\text{diag}(\nu_1)}\} \cup \text{R}_{\text{P}0}$, and prioritize three types of derived generators. First, we add back the basis generators $\{t'_{e_i}\}_{i=2}^n$ and $\{r'_{\text{diag}(\nu_i)}\}_{i=2}^n$, so we get back the generating set $\text{T}_0 \cup \text{R}_{\text{N}0} \cup \text{R}_{\text{P}0}$ from (9.59). This restores the complete sets of translations, negations, and permutations—the three independent pillars generating $\text{ISO}(\mathbb{Z}^n)$. The other two types of derived generators, called *circulators* and *synchronizers*, are inspired by biologically innate preference for *periodicity* and *synchronization* [84, 87, 88].

Definition 9.11. *Let $S = \{s_1, \dots, s_k\}$ be a minimal generating set. A circulator of S with period α is: s^α for some $s \in S$ and $\alpha \in \mathbb{Z}_{>0}$. (Consider group action on X and any $x \in X$: the orbit $\langle s^\alpha \rangle x$ consists of periodic points from $\langle s \rangle x$.) If $\langle S \rangle$ is Abelian, the synchronizer of S is: $s_k \circ \dots \circ s_1$.*

We denote the set of all circulators of T_0 with a fixed period α by $\text{T}_0^\alpha := \{t'_{\alpha e_i}\}_{i=1}^n$. Inspecting circulators of $\text{R}_{\text{N}0}$ and $\text{R}_{\text{P}0}$ does not yield new generators, since for any $s \in \text{R}_{\text{N}0} \cup \text{R}_{\text{P}0}$, $s^2 = \text{id}$. The synchronizers of T_0 and $\text{R}_{\text{N}0}$ are t'_1 and r'_{-I} ($\langle \text{R}_{\text{P}0} \rangle$ is not Abelian). Adding these circulators and synchronizers to $\text{T}_0 \cup \text{R}_{\text{N}0} \cup \text{R}_{\text{P}0}$ yields the generating set:

$$S_+^* := \text{T}_0 \cup \text{T}_0^2 \cup \dots \cup \text{T}_0^\tau \cup \{t'_1\} \cup \text{R}_{\text{N}0} \cup \{r'_{-I}\} \cup \text{R}_{\text{P}0}, \quad (9.62)$$

where τ denotes an upper bound on period exploration. Note that $|S_+^*| = \tau n + 1 + n + 1 + (n - 1) = (\tau + 2)n + 1$. In light of real applications, we can use this generating set to generate an abstraction semiuniverse for automatic music concept learning.

9.4 RESTRICTION TO FINITE SUBSPACES

Computers have to work with finite spaces for finite execution time. If the underlying space X is finite, then there is no issue. However, if X is infinite (but still discrete) like \mathbb{Z}^n ,

we have to consider a finite subspace of X in practice. In this case, we must be careful about both what an abstraction of a subspace means and what potential problems might occur.

Definition 9.12. *Let X be a set and \mathcal{P} be an abstraction of X . For any $Y \subseteq X$, the restriction of \mathcal{P} to Y is an abstraction of Y given by $\mathcal{P}|_Y := \{P \cap Y \mid P \in \mathcal{P}\} \setminus \{\emptyset\}$.*

Remark 9.11. *Unless otherwise stated, the term “an abstraction of a subspace” means an abstraction of the ambient space restricted to that subspace. Under this definition, we need extra caution when computing an abstraction of a subspace.*

Let X be a set, and $H \leq \mathbf{F}(X)$ be a subgroup of the transformation group of X . For any $Y \subseteq X$, according to Definition 9.12, the correct way of generating the abstraction of the subspace Y from H is:

$$\pi(H)|_Y = (X/H)|_Y = \{Hx \cap Y \mid x \in X\} \setminus \{\emptyset\}. \quad (9.63)$$

A risky way of computing the abstraction of the subspace Y is by thinking only in Y while forgetting the ambient space X . The risk here is to get a partition of Y , denoted \mathcal{R}_Y^H , which is strictly finer than $\pi(H)|_Y = \mathcal{R}_X^H|_Y$. In other words, there are possibly cells in \mathcal{R}_Y^H that should be merged but are not if they are connected via points outside the subspace Y . For instance, consider $X = \mathbb{Z}^2$, $Y = \{(0, 0), (1, 0), (0, 1), (1, 1)\} \subseteq X$, and the subgroup $H = \langle \{t'_1, r'_{-I}\} \rangle \leq \text{ISO}(\mathbb{Z}^2) \leq \mathbf{F}(\mathbb{Z}^2)$. Let \mathcal{R}_Y^H be the abstraction of Y obtained by running the induction algorithm on Y (instead of X) in the bottom-up approach. One can check:

$$\mathcal{R}_Y^H = \{ \{(0, 0), (1, 1)\}, \{(0, 1)\}, \{(1, 0)\} \}; \quad (9.64)$$

$$\pi(H)|_Y = \{ \{(0, 0), (1, 1)\}, \{(0, 1), (1, 0)\} \}. \quad (9.65)$$

The two points $(1, 0)$ and $(0, 1)$ should be in one cell since $(1, 0) \xrightarrow{r'_{-I}} (-1, 0) \xrightarrow{t'_1} (0, 1)$, but are not in \mathcal{R}_Y^H since the via-point $(-1, 0) \notin Y$. In general, the risk is present if we compute an abstraction of a subspace Y from other abstractions of Y or from orbit tracing.

However, for computational reasons, we want to forget the ambient space X ! In particular, the risky way is the only practical way if X is infinite and Y is finite, since it is not realistic to identify all orbits in an infinite space. This suggests that we take the risk to generate \mathcal{R}_Y^H as the first step, and rectify the result in a second step to merge cells that are missed in the first step. As a result, we introduce a technique called “expand-and-restrict”.

9.4.1 Expand-and-Restrict

“Expand-and-restrict” is an empirical technique which first expands the subspace and then restricts it back, i.e. to compute $\mathcal{R}_{Y_+}^H|_Y$ for some finite subspace Y_+ such that $Y \subset Y_+ \subset X$. The expansion Y_+ takes more via-points into consideration, so it helps merge cells that are missed in \mathcal{R}_Y^H . In practice, we carry out this technique gradually in a sequential manner, which is similar to what we did in enlarging a minimal generating set (cf. Section 9.3.4). Given an infinite space X and a finite subspace $Y \subset X$, we first construct a filtration

$$Y = Y_{+0} \subset Y_{+1} \subset Y_{+2} \subset \cdots \subset X \quad \text{where } Y_{+k} \text{ is finite } \forall k \in \mathbb{Z}_{\geq 0} \text{ and } \bigcup_{k=0}^{\infty} Y_{+k} = X. \quad (9.66)$$

We then start a search process for a good expansion Y_{+k} . More specifically, we iteratively compute $\mathcal{R}_{Y_{+k}}^H|_Y$ for expansion factors $k = 0, 1, 2, \dots$ until the results reach a consensus among consecutive iterations. To determine a consensus, theoretically, we need to find the smallest k such that $\mathcal{R}_{Y_{+k}}^H|_Y = \mathcal{R}_{Y_{+k'}}^H|_Y$ for all $k' > k$, which requires an endless search. In practice, we stop the search whenever $\mathcal{R}_{Y_{+k}}^H|_Y = \mathcal{R}_{Y_{+(k+1)}}^H|_Y = \cdots = \mathcal{R}_{Y_{+(k+\Delta k)}}^H|_Y$ for some positive integer Δk . We call this an *early stop*, whose resulting abstraction $\mathcal{R}_{Y_{+k}}^H|_Y$ is an empirical approximation of the true abstraction $\pi(H)|_Y$. Note that without early stopping, we will have the correct result $\pi(H)|_Y = \mathcal{R}_X^H|_Y$ in the limit of this infinite search process. Therefore, even in cases where the space X is finite, if X is much larger than the subspace Y , this empirical search can be more efficient than computing $\pi(H)|_Y$ directly, since earlier search iterations will be extremely cheap and if an early stop happens early there is a win.

9.4.2 An Implementation Example

We give an example to illustrate some implementation details on generating abstractions of a finite subspace. In this example, we consider finite subspaces of $X = \mathbb{Z}^n$ to be the centered hypercubes of the form $Y = \mathbb{Z}_{[-b,b]}^n$ where $\mathbb{Z}_{[-b,b]} := \mathbb{Z} \cap [-b, b]$ and $b > 0$ is finite. To construct an abstraction semiuniverse for such a finite hypercube, we adopt the bottom-up approach and pick the generating set to be S_+^* defined in (9.62). Taking S_+^* and $\mathbb{Z}_{[-b,b]}^n$ as inputs, we run the induction algorithm, where both Algorithm 9.1 (for base cases) and Algorithm 9.2 (for the induction step) are run on the finite subspace $\mathbb{Z}_{[-b,b]}^n$ instead of the infinite space \mathbb{Z}^n . This is the first step which gives abstractions $\mathcal{R}_{\mathbb{Z}_{[-b,b]}^n}^{(S)}$ for $S \subseteq S_+^*$.

As mentioned earlier, for every $S \subseteq S_+^*$, the correct abstraction should be $\mathcal{R}_{\mathbb{Z}_{[-b,b]}^n}^{(S)}|_{\mathbb{Z}_{[-b,b]}^n}$ which is generally not equal to $\mathcal{R}_{\mathbb{Z}_{[-b,b]}^n}^{(S)}$. So, we run the “expand-and-restrict” technique as the second step. We first construct a filtration: let $Y_{+k} = \mathbb{Z}_{[-b-k, b+k]}^n$ be a finite expansion of

$Y = \mathbb{Z}_{[-b,b]}^n$, then it is clear that $Y = Y_{+0} \subset Y_{+1} \subset Y_{+2} \subset \dots \subset X$ and $\cup_{k=0}^{\infty} Y_{+k} = X = \mathbb{Z}^n$. We then start the empirical search process and set $\Delta k = 1$ (the most greedy search). This means we will stop the search whenever $\mathcal{R}_{Y_{+k}}^{(S)}|_Y = \mathcal{R}_{Y_{+(k+1)}}^{(S)}|_Y$, and return the abstraction $\mathcal{R}_{Y_{+k}}^{(S)}|_Y = \mathcal{R}_{\mathbb{Z}_{[-b-k,b+k]}^n}^{(S)}|_{\mathbb{Z}_{[-b,b]}^n}$ as the final result to approximate $\mathcal{R}_{\mathbb{Z}^n}^{(S)}|_{\mathbb{Z}_{[-b,b]}^n} = \Pi(S)|_{\mathbb{Z}_{[-b,b]}^n}$.

There are three additional implementation tricks that are special to this example. The first trick applies to cases where the subspace $\mathbb{Z}_{[-b,b]}^n$ is large, i.e. a large b . In this case, every search iteration in the “expand-and-restrict” technique is expensive and gets more expensive as the search goes. However, for the generating set S_+^* specifically, it is typical to have $b \gg \tau$ so as to reveal strong periodic patterns. Thus, we run the entire two-step abstraction generating process for $\mathbb{Z}_{[-\tau,\tau]}^n$ instead of $\mathbb{Z}_{[-b,b]}^n$, pretending $\mathbb{Z}_{[-\tau,\tau]}^n$ is the subspace that we want to abstract. This yields a much faster abstraction process since $\mathbb{Z}_{[-\tau,\tau]}^n$ is much smaller than $\mathbb{Z}_{[-b,b]}^n$. The result is an abstraction $\mathcal{R}_{\mathbb{Z}_{[-\tau-k,\tau+k]}^n}^{(S)}|_{\mathbb{Z}_{[-\tau,\tau]}^n}$ for some expansion factor k . We reuse this same k and compute $\mathcal{R}_{\mathbb{Z}_{[-b-k,b+k]}^n}^{(S)}|_{\mathbb{Z}_{[-b,b]}^n}$ as the final result, which is the only expensive computation. Note that this trick adds an additional empirical approximation, assuming that the same expansion factor k works for both small and large subspaces. While we have not yet found a theoretical guarantee for this assumption, this trick works well in practice, and provides huge computational savings.

Note: for some generating subsets $S \subseteq S_+^*$, we can prove (so no approximations) that the expansion factor $k = 0$ (no need to expand) or 1. Although this provides theoretical guarantees in certain cases, the tricks used in the current proofs are case-by-case depending on the chosen generators. Thus, before we find a universal way of proving things, we prefer empirical strategies—like the above search process—which work universally in any event.

The second trick considers the subspace to be any general hypercube in \mathbb{Z}^n , which is not necessarily square or centered. The trick here is simply to find a minimum centered square hypercube containing the subspace. If the ambient space X happens to be “spatially stationary”—the absolute location of each element in the space is not important but only their relative position matters (e.g. the space of music pitches)—then we find a minimum square hypercube containing the subspace and center it via a translation. Centering is very important and specific to the chosen generating set S_+^* . This is because S_+^* contains only pure translations and pure rotations; and centering square hypercubes makes pure rotations safe: no rotation maps a point in $\mathbb{Z}_{[-b,b]}^n$ outside (one can check that for any $r'_A \in \mathbf{R}(\mathbb{Z}^n)$, $r'_A(\mathbb{Z}_{[-b,b]}^n) = \mathbb{Z}_{[-b,b]}^n$). In practice, centering dramatically decreases the number of miss-merged cells, and makes it safe to choose small Δk for early stopping. This explains why we only consider subspaces of the form $\mathbb{Z}_{[-b,b]}^n$ in the first place, and boldly choose $\Delta k = 1$.

The third trick considers a quick-and-dirty pruning of duplicates in generating a family of

partitions, leaving room for larger-period explorations. Without this trick, to generate the partition family $\Pi(2^{S_+^*})$, we need $|2^{S_+^*}| = 2^{(\tau+2)n+1} = O(2^\tau)$ computations, which hinders exploration on period τ . However, S_+^* is not minimal, so $|\Pi(2^{S_+^*})| < |2^{S_+^*}|$, suggesting many computations are not needed since they yield the same abstraction. We focus on circulators, where we exclude computations on those $S \in 2^{S_+^*}$ containing multiple periods. This reduces the number of computations to $(2^{3n+1} - 2^{2n+1})\tau + 2^{2n+1} = O(\tau)$.

A real run on the subspace $\mathbb{Z}_{[-12,12]}^4$ and $\tau = 4$ computes 31232 partitions, during which all search processes in “expand-and-restrict” end in at most three iterations. This means in this experiment we only need to expand the subspace by $k = 0$ or 1 for all abstractions in the family.

Lastly, we briefly mention the task of completing a global hierarchy on an abstraction family \mathfrak{P}_Y . A brute-force algorithm makes $O(|\mathfrak{P}_Y|^2)$ comparisons, determining the relation (\preceq or incomparable) for every unordered pair of partitions $\mathcal{P}, \mathcal{Q} \in \mathfrak{P}_Y$. Locally, we run a subroutine `GetRelation(P,Q)` implemented via the contingency table [78] whenever we want to query a pair of partitions. Globally, we use two properties to reduce the number of calls to `GetRelation(P,Q)`: 1) transitivity: for any $\mathcal{P}, \mathcal{P}', \mathcal{P}'' \in \mathfrak{P}_Y$, $\mathcal{P} \preceq \mathcal{P}'$ and $\mathcal{P}' \preceq \mathcal{P}''$ implies $\mathcal{P} \preceq \mathcal{P}''$; 2) dualities in Theorem 6.3. The final output of our abstraction process is a directed acyclic graph of the abstraction (semi)universe $\Pi(2^{S_+^*})$. Similar to the first trick above, in practice it suffices to complete the hierarchy for smaller subspaces like $\mathbb{Z}_{[-\tau,\tau]}^n$, assuming the same hierarchy holds for the actual subspace under consideration.

Chapter 10: Information Lattice Learning Phase II: Probabilistic Rule Learning

We present algorithmic guidelines for probabilistic rule learning, the second phase of our two-phase Information Lattice Learning (ILL). The essence of this Phase II learning is to couple abstraction hierarchy (from Phase I) with data statistics, and the resulting output of the rule learning algorithm is a rule trace, i.e. an ordered sequence of probabilistic rules recording a complete rule-learning path. Any learned rule trace is also a final ACL output which reveals structured, independent, and human-interpretable insights of the input data.

In this chapter, Section 10.1 first introduces the vanilla version of a self-learning loop, which is an automated learning cycle between two learning agents: a “teacher” that extracts rules and a “student” that applies rules. For each of the two learning agents respectively, Section 10.2 further presents a more advanced teacher, which intelligently matches abstraction contexts to adaptively produce both context-free and context-dependent rules; Section 10.3 further presents a more advanced student, which intelligently scrutinizes any given rules to elastically apply only a self-selected subset of them while breaking the others.

10.1 THE VANILLA “TEACHER \rightleftharpoons STUDENT” LOOP

Informally speaking, probabilistic rule learning is accomplished in a procedural way, by each time learning a new and independent rule which from an unadopted perspective, reveals a yet-not-discovered insight of the data. By a new and independent rule, we mean a rule that is not implied by all previously learned rule, i.e. a rule that is “perpendicular” to all previous rules. This is reminiscent to our own learning experience, say in a semester-long class, where in every class through a semester, we always expect to learn something new, something we haven’t seen before, and something that is “perpendicular” to our prior knowledge.

Formally, probabilistic rule learning is achieved by a so-called *self-learning loop*. The loop adopts a teacher \rightleftharpoons student architecture whose main idea is *learning by comparison*. This setting finds its prototype from a typical human pedagogical scenario where a teacher guides a student to a designated goal through iterative exercises and feedback (Figure 10.1: left). As a computational counterpart (Figure 10.1: right), in our self-learning loop, the teacher is a discriminative model and the student is a generative model. The two learning agents cooperate with each other iteratively, with the teacher extracting rules for the student and the student applying rules to generate data for more feedback (i.e. the rules) from the teacher.

We first present an overview of our self-learning loop (Figure 10.1: right), showing the

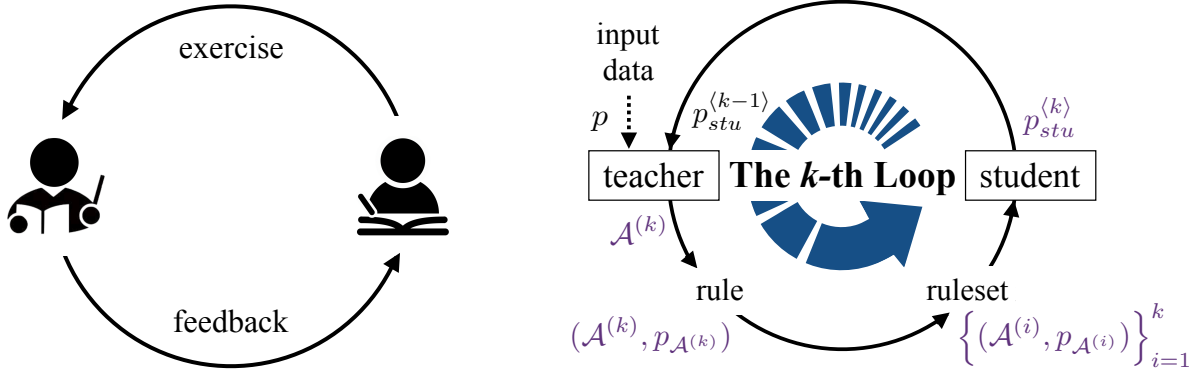


Figure 10.1: The teacher \rightleftharpoons student architecture in a typical human pedagogical scenario (left) and in our self-learning loop, a computational counterpart (right).

big picture. At the beginning of the k th loop (or iteration), the teacher takes two inputs, namely the input data distribution p and the student’s estimation of it $p_{stu}^{(k-1)}$ by the end of the previous loop. These two data distributions are treated as the target probabilistic model and the student’s latest probabilistic model, respectively. The teacher tries to identify one abstraction $\mathcal{A}^{(k)}$ under which the two probabilistic models exhibit the largest difference. This particular abstraction $\mathcal{A}^{(k)}$ will be made into a rule $(\mathcal{A}^{(k)}, p_{\mathcal{A}^{(k)}})$ and inserted into a ruleset. Then the student takes as input the augmented ruleset $\{(\mathcal{A}^{(i)}, p_{\mathcal{A}^{(i)}})\}_{i=1}^k$ to update its probabilistic model into $p_{stu}^{(k)}$ which marks the end of the k th loop. This iterative process looks very similar to a generative adversarial network [27], however, rather than using neural networks, the implementations of both the teacher and the student are largely different in order to maintain a transparency of the entire loop, and thus, a transparency of the entire learning process. We will next elaborate our vanilla implementations of the teacher and the student, respectively.

10.1.1 The Teacher: a Discriminative Model

Informally speaking, the teacher, as a discriminative model, aims to find the abstraction which reveals the largest statistical difference between the student’s probabilistic model and the target probabilistic model (i.e. the data statistics). This word description is formalized into the following optimization problem, with the optimization variable being an abstraction $\mathcal{A} \in \mathfrak{P}_X$ drawn from an abstraction hierarchy generated in ILL Phase I.

$$\underset{\mathcal{A} \in \mathfrak{P}_X}{\text{maximize}} \quad D_{KL} \left(p_{\mathcal{A},stu}^{(k-1)} \parallel p_{\mathcal{A}} \right) \quad (10.1)$$

$$\text{subject to} \quad \mathcal{A} \notin \mathfrak{P}^{(k-1)} \quad (10.2)$$

$$H(p_{\mathcal{A}}) \leq \delta_k \quad (10.3)$$

The objective function (10.1) is the Kullback-Leibler (KL) divergence (relative entropy) between the student’s latest probabilistic model and the target probabilistic model but after projecting both onto an abstraction space. It is important to notice that what are compared here are not the two probabilistic models $p_{stu}^{(k-1)}$ and p , but their *projections* onto an abstraction space $p_{\mathcal{A},stu}^{(k-1)}$ and $p_{\mathcal{A}}$. The KL divergence quantifies the statistical difference, which is picked among other possibilities (e.g. ℓ_1 norm) due to its nice interpretation as the information gain from the student to the target. In other words, the KL divergence shows the unidirectional gap for the student to catch up on, since it is the student who tries to mimic the target, but not the other way around. Given any probability distribution q_X of the input data space X , its projection onto an abstraction \mathcal{A} of X can be easily inferred by aggregating probabilities within each concept (i.e. partition cell):

$$q_{\mathcal{A}}(C) = \sum_{x \in C} q_X(x) \quad \text{for any } \mathcal{A} \in \mathfrak{P}_X \text{ and any } C \in \mathcal{A}. \quad (10.4)$$

The constraint (10.2) restricts attention to only new and independent rules. More specifically, the feasible set of candidate abstractions explicitly excludes all previously learned rule abstractions, where in this vanilla version $\mathfrak{P}^{(k-1)} := \{\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(k-1)}\}$ denotes the set of all $k - 1$ rule abstractions extracted from the previous iterations. Later in Section 10.2, for a more advanced teacher, the exclusion set $\mathfrak{P}^{(k-1)}$ will expand to abstractions from not only all previously learned rules but also rules that can be implied from all previous rules; further this implication will include both conceptual implications (from rule hierarchy) and informational implication (from statistics), both of which will be detailed in Section 10.2.

The constraint (10.3) imposes a regularity condition for the expected rule to be extracted. More specifically, we expect a “good” rule to reveal hidden probability concentrations in the original data distribution, where the concentration is measured by setting the Shannon entropy H below a certain threshold δ_k , essentially driving a probabilistic rule towards a deterministic rule if possible. It is worth noting that this regularity (or concentration) constraint caps the so-called *entropic difficulty* of a probabilistic rule, depicting how difficult it is for a human to memorize the rule. Intuitively, the larger the entropy is, the more

difficult the rule is for people to memorize; or to say it another way, rules that are more deterministic are easier for people to memorize [89]. The threshold δ_k is a hyper-parameter for the k th self-learning loop which is to be pre-selected before solving the optimization problem, and δ_k can vary from iteration to iteration to provide one way of personalizing the resulting rule trace. Later in Section 10.2, we will further discuss strategies for setting this hyper-parameter, together with many others, to control the rule learning pace.

Lastly, let \mathcal{A}^* denote the solution to the optimization problem (10.1), then the abstraction extracted in the k th iteration $\mathcal{A}^{(k)} = \mathcal{A}^*$. In some variants of teacher’s optimization problem, the regularity constraint (10.3) is integrated into the objective function (10.1) to form a scoring function that measures differentiability and regularity as well as their trade-off at the same time. Yet, it is clear that these variants share the same idea here in designing the teacher.

10.1.2 The Student: a Generative Model

Opposite to what the teacher does which extracts probabilistic rules, the student applies probabilistic rules, which is also known as the *rule realization* problem. A rule realization problem takes as inputs multiple probabilistic rules $(\mathcal{A}^{(1)}, p_{\mathcal{A}^{(1)}}), (\mathcal{A}^{(2)}, p_{\mathcal{A}^{(2)}}), \dots$, and looks for a data distribution p such that its projections onto $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \dots$ agree with $p_{\mathcal{A}^{(1)}}, p_{\mathcal{A}^{(2)}}, \dots$. Therefore, rule realization is essentially the converse of projecting a data distribution onto abstraction spaces. Unlike the projection problem $p \rightarrow p_{\mathcal{A}}$ where $p_{\mathcal{A}}$ is uniquely determined, the solution to a rule realization problem $p_{\mathcal{A}^{(1)}}, p_{\mathcal{A}^{(2)}}, \dots \rightarrow p$ is not necessarily unique. Intuitively, if the number of rules is small, the solution set tends to be large (for instance, if there is only one rule, the probability mass within each concept is totally undecided as long as the sum matches the probability of the concept), and with more and more rules added into the rule set, the solution set gets smaller and smaller (more constrained by the increasing number of rules), but still comprises multiple solutions most of the time. Then, whenever there are two distinct data distributions p and p' such that $p_{\mathcal{A}^{(i)}} = p'_{\mathcal{A}^{(i)}}$ for all i , which one do we prefer?

Informally speaking, the student, as a generative model, aims to find the most creative probabilistic model, one that satisfies all the rules while at the same time enables novelty. In other words, among all solutions to a rule realization problem, we prefer the one that is most novel. Following a general definition, we judge creativity by quality and novelty [90]. In our case for the student, we judge quality by how well the student obeys the rules (so the exact solutions to a rule realization problem all have the highest quality); we judge novelty by how random the student is (a novel solution encourages all possibilities with

equal probability). This word description is formalized into the following maximum entropy optimization problem [91], with the optimization variable being the student’s probabilistic model (Δ^n denotes an n -dimensional probability simplex), i.e. the data distribution of the raw input space from the student’s perspective.

$$\underset{p_{stu}^{(k)} \in \Delta^{|X|}}{\text{maximize}} \quad S_q \left(p_{stu}^{(k)} \right) := (1 - \|p_{stu}^{(k)}\|_q^q) / (q - 1) \quad (10.5)$$

$$\text{subject to} \quad A^{(i)} p_{stu}^{(k)} = p_{\mathcal{A}^{(i)}}, \quad i = 1, \dots, k \quad (10.6)$$

The objective function (10.5) is the q th order Tsallis entropy of the student’s probabilistic model. Tsallis entropy is a generalization of the Shannon entropy ($q \rightarrow 1$) and Gini impurity ($q = 2$), which measures the randomness of student’s probabilistic model, a surrogate of the student’s novelty. By maximizing the Tsallis entropy, we want the student to be as random (novel) as possible as long as the rules are satisfied which are encoded in the constraints.

The constraint (10.6) encodes the rule requirements, comprising exactly k probabilistic rules that have been extracted by the teacher up until the current k th iteration. Each rule requirement is coded as a linear equality constraint, where the matrix $A^{(i)}$ is the partition matrix of the abstraction $\mathcal{A}^{(i)}$. One can verify that every linear equality constraint literally says that the projection of the student’s probabilistic model $p_{stu}^{(k)}$ onto the abstraction $\mathcal{A}^{(i)}$ must agree with its probability distribution $p_{\mathcal{A}^{(i)}}$ (over the concepts).

We mention two special cases: one for $q = 2$ (Gini impurity); the other for $q \rightarrow 1$ (the Shannon entropy). The former has computational simplicity, and so will be the main version implemented; the latter has better conceptual interpretation, which reveals a closer connection to Shannon’s information lattice.

When $q = 2$ in the Tsallis entropy, the objective becomes minimizing the ℓ_2 -norm of the student’s probabilistic model; hence, the entire optimization problem becomes a standard linear least-squares problem.

When $q \rightarrow 1$ in the Tsallis entropy, the objective function $S_q(p_{stu}^{(k)}) \rightarrow H(p_{stu}^{(k)})$, and the entire optimization problem can be equivalently rewritten as an inference problem in Shannon’s information lattice. To see this more explicitly, and for notational ease, we first let $x = p_{stu}^{(k)}$ and $y^i = p_{\mathcal{A}^{(i)}}$ denote the probability distribution of the raw input space and the probability distribution of the i th abstraction space, respectively. We slightly overload the notations and let \mathbf{x} and \mathbf{y}^i be the information elements that represent the probability space $(X, \Sigma(I), x)$ and $(X, \Sigma(A^{(i)}), y^i)$, respectively. In this notation, the sample space X is the raw input space and, for a partition matrix P , $\Sigma(P)$ denotes the σ -algebra generated by the

partition represented by P (so $\Sigma(I)$ denotes the σ -algebra generated by the finest partition). Under this setting, the equality constraint $A^{(i)}x = y^i$ becomes $H(\mathbf{y}^i|\mathbf{x}) = 0$, i.e. \mathbf{y}^i is an abstraction of \mathbf{x} as information elements or $\mathbf{y}^i \leq \mathbf{x}$ by Shannon’s notation [30]. Therefore, the student’s optimization problem for rule realization can be rewritten as follows:

$$\text{maximize } H(\mathbf{x}) \tag{10.7}$$

$$\text{subject to } H(\mathbf{y}^i|\mathbf{x}) = 0, \quad i = 1, \dots, k \tag{10.8}$$

In words, this means that we want to find an information lattice for the student (used as its mental model) such that it agrees on all k abstractions from the information lattice for the data set and meanwhile achieves the largest randomness in the input space. As a result, a good student memorizes high-level principles—rules in terms of high-level abstractions and their statistical patterns—in data generation rather than the data or the data distribution itself. Indeed the student is encouraged to be as creative as possible as long as the high-level principles are satisfied.

10.2 THE ADAPTIVE TEACHER: RULE CONTEXT-MATCHING

Built on top of the vanilla teacher (Section 10.1.1), this section presents an adaptive teacher. This more advanced teacher upgrades the vanilla version by further strengthening the following two contextual structures of probabilistic rules, namely *informational context* and *temporal context*. The more advanced teacher is adaptive in adjusting the learning pace accordingly to the informational context, and is also adaptive in matching the temporal context for time-series data.

10.2.1 Informational Context

We exploit, at a deeper level, the hierarchy of information lattices (i.e. abstraction universes equipped with probability measures). In particular, we leverage *conceptual hierarchy* that is pre-determined by abstraction hierarchy, and further infer *informational hierarchy* that is post-implied from an information-theoretic perspective. More specifically, the former is directly inherited from the partition-lattice structure in ILL Phase I, and the latter is inferred from data statistics. Accordingly, a probabilistic rule can be *implied* from other probabilistic rule(s) in the following two ways.

- a) We say a probabilistic rule $(\mathcal{A}, p_{\mathcal{A}})$ is *conceptually implied* from another probabilistic rule $(\mathcal{A}', p_{\mathcal{A}'})$, if the two abstractions satisfy $\mathcal{A} \preceq \mathcal{A}'$. In this case, given $p_{\mathcal{A}'}$, the

probabilistic pattern $p_{\mathcal{A}}$ of any of its higher-level abstractions (i.e. coarser partitions) is completely determined *a priori* without learning.

- b) We say a probabilistic rule $(\mathcal{A}, p_{\mathcal{A}})$ is *informationally implied* from another set of probabilistic rules $\{\mathcal{A}^{(i)}, p_{\mathcal{A}^{(i)}}\}_{i=1}^k$ if the probabilistic model q that realizes the rule set also coincides with $p_{\mathcal{A}}$, i.e. $q_{\mathcal{A}} = p_{\mathcal{A}}$, or most often in a loose sense $q_{\mathcal{A}} \approx p_{\mathcal{A}}$ represented by $D_{KL}(q_{\mathcal{A}} \parallel p_{\mathcal{A}}) \leq \gamma$ for some threshold γ . In this case, given the rule set $\{\mathcal{A}^{(i)}, p_{\mathcal{A}^{(i)}}\}_{i=1}^k$, the probabilistic pattern on \mathcal{A} is already close to the target even without learning.

One might question the necessity of conceptual hierarchy since it can be implied in the informational hierarchy. The answer is yes in principle, but no in practice. The main difference is that conceptual hierarchy is pre-computed over the entire abstraction universe before the loop, which is global, precise, and trace independent. On the contrary, informational hierarchy is trace specific and loose, due to tolerance γ and the precision of the optimization solver. As a result, informational hierarchy alone tends to lose the big picture and require more post-hoc interpretations, and is unstable in practice.

The abstraction of an implied probabilistic rule, whether conceptual or informational, can be excluded from the feasible set of candidate abstractions when solving the teacher’s optimization problem. This is because explicitly learning these implied rules does not provide new insights into the data. So, the set $\mathfrak{P}^{(k-1)}$ of learned rule abstractions can be further expanded to include abstractions from implied rules. This upgrades a vanilla teacher’s optimization problem into the following one for a more advanced teacher.

$$\begin{aligned}
 & \underset{\mathcal{A} \in \mathfrak{P}_X}{\text{maximize}} && D_{KL} \left(p_{\mathcal{A},stu}^{(k-1)} \parallel p_{\mathcal{A}} \right) && (10.9) \\
 & \text{subject to} && \mathcal{A} \notin \mathfrak{C}^{(k-1)} := \{ \mathcal{A} \mid \mathcal{P}_{\mathcal{A}} \preceq \mathcal{P}_{\mathcal{A}'}, \mathcal{A}' \in \mathfrak{P}^{(k-1)} \} && \text{(conceptual-hierarchy filter)} \\
 & && \mathcal{A} \notin \mathfrak{J}^{(k-1)} := \{ \mathcal{A} \mid D_{KL}(p_{\mathcal{A},stu}^{(k-1)} \parallel p_{\mathcal{A}}) < \gamma_k \} && \text{(informational-hierarchy filter)} \\
 & && H(p_{\mathcal{A}}) \leq \delta_k && \text{(regularity condition)}
 \end{aligned}$$

Beyond their benefits in revealing inter-relational insights among distributed abstractions and representations, the hierarchical filters (both conceptual and informational) added in a more advanced teacher’s optimization problem are computationally beneficial in pruning hierarchically implied rules, so that solving this optimization problem is made more efficient.

The two hyper-parameters γ_k and δ_k in a more advanced teacher’s optimization problem (10.9) are used collectively to control the rule-learning pace, and thus, to personalize a

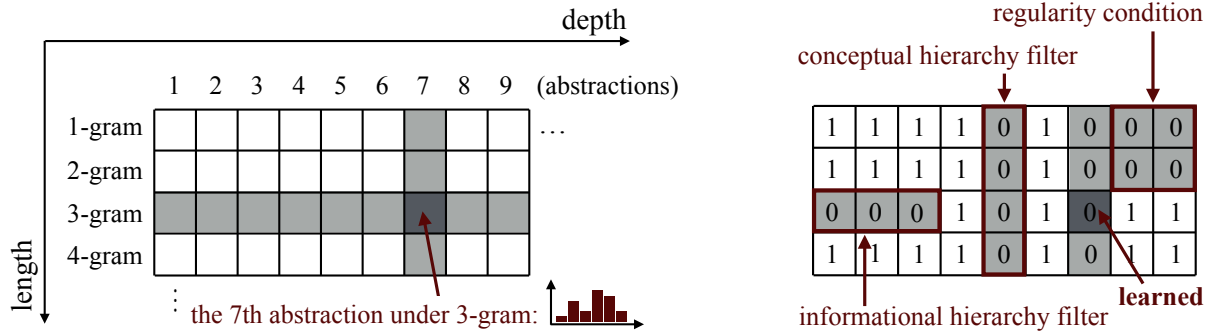


Figure 10.2: Two-dimensional memory (left): the length axis enumerates n -gram orders; the depth axis enumerates abstractions; and every cell stores the probabilistic pattern of an (un)conditional rule. Memory mask (right): 0 marks the removal of the corresponding cell from the feasible set of candidate abstractions, which is caused by a hierarchical filter or the regularity condition or (contradictory) duplication.

rule trace. We pre-select the threshold γ_k before the k th loop to express a user’s *satisfaction level*: a smaller γ_k signifies a meticulous user who is harder to satisfy; the threshold δ_k upper bounds the *entropic difficulty* of the rules, and can be adaptively adjusted throughout the loops: roughly speaking, it starts from a small value (easy rules first), and auto-increases whenever the feasible set is empty. In summary, the high-level strategy here is to gradually increase the difficulty (measured by δ_k) when mastering the current level (where the extent of mastering is measured by γ_k).

10.2.2 Temporal Context

For time-series data where every data sample is organized as a sequence of data points instead of an individual data point, we can simply execute the self-learning loop on a language model, say n -grams, to extract both unconditional probabilistic rules and conditional ones. This requires a more advanced teacher that is able to adaptively extract rules in a multi-abstraction multi- n -gram language model. More specifically, we consider a continuous range of n -grams—also called variable memory (or temporal context with variable lengths)—and adaptively pick the optimal order n based on a balance made among several criteria that will be introduced soon. It is important to notice that every n -gram is built on top of multiple high-level abstraction spaces that are generally much smaller than the original raw input space. Therefore, this opens the opportunity for long-term memories without exhausting machine memory, while at the same time, effectively avoids overfitting.

Two-Dimensional Memory. Considering a continuous range of n -grams, say $n \in N = \{1, 2, 3, \dots\}$, the abstraction universe adds another dimension, forming a two-dimensional

memory ($N \times \mathfrak{P}_X$)—*length* versus *depth*—for the language model (Figure 10.2: left). The length axis enumerates n -gram orders, with a longer memory corresponding to a larger n ; the depth axis enumerates abstractions, with a deeper memory corresponding to a higher-level abstraction. Every cell in the memory is indexed by two coordinates (n, \mathcal{A}) , referring to the abstraction \mathcal{A} under the specific n -gram, and stores the corresponding probability distribution. As a result, the teacher’s rule extraction task involves picking the most appropriate abstraction under the most applicable n -gram, which extends the space of the teacher’s optimization problem (10.9) from \mathfrak{P}_X to $N \times \mathfrak{P}_X$. Accordingly, the constraints jointly forge a mask on top of the 2D memory (Figure 10.2: right).

Criteria and Balance. We propose three criteria to extract rules from the 2D memory: confidence, regularity, and efficacy. *Confidence* is quantified by empirical counts: the more relevant examples one sees in a data set, the more confident. *Regularity* is quantified by the Shannon entropy of the rule’s probability distribution: a rule is easier to memorize if it has a lower entropy [89]. *Efficacy* is inversely quantified by the statistical difference between the student’s probabilistic model and the target probabilistic model: a rule is more effective if it exposes a larger difference. There are trade-offs among these criteria. For instance, a lower-level abstraction is usually more effective since it records more low-level details and normally reflects a larger statistical difference, but it is also unlikely to be regular, thus, is harder to memorize and to generalize. Also an abstraction under a higher-order n -gram may be both regular and effective, but the number of examples that match the long-term conditionals is likely to be small, reducing confidence.

Adaptive Selection: Follow the (Bayesian) Surprise. The teacher’s optimization problem (10.9) explicitly expresses the efficacy factor in the objective, and the regularity condition as the last constraint. To further incorporate confidence, we cast a rule’s probability distribution $p_{\mathcal{A}}$ in a Bayesian framework rather than a purely empirical framework. We first assume the student’s belief with respect to an abstraction \mathcal{A} follows a Dirichlet distribution whose expectation is the student’s probabilistic model. In the k th iteration of the self-learning loop, we set the student’s prior belief as the Dirichlet distribution parameterized by the student’s latest probabilistic model:

$$\text{prior}_{\mathcal{A},stu} \sim \text{Dir} \left(c \cdot p_{\mathcal{A},stu}^{(k-1)} \right), \quad (10.10)$$

where $c > 0$ denotes the strength of the prior. From the target distribution p (or its empirical estimation), the teacher inspects $p_{\mathcal{A}}$ associated with the abstraction \mathcal{A} and the relevant n -

gram, and further computes the student’s posterior belief if \mathcal{A} were selected for the rule:

$$\text{posterior}_{\mathcal{A},stu} \sim \text{Dir} \left(p_{\mathcal{A}} + c \cdot p_{\mathcal{A},stu}^{\langle k-1 \rangle} \right). \quad (10.11)$$

The concentration parameters of the Dirichlet posterior show the balance between the target distribution and the prior. If the confidence level for $p_{\mathcal{A}}$ is low, the posterior will be smoothed more by the prior, de-emphasizing the target distribution $p_{\mathcal{A}}$. If we compute a rectified target distribution $\hat{p}_{\mathcal{A}} \propto \left(p_{\mathcal{A}} + c \cdot p_{\mathcal{A},stu}^{\langle k-1 \rangle} \right)$ in the objective of the optimization problem (10.9), then

$$D_{KL} \left(p_{\mathcal{A},stu}^{\langle k-1 \rangle} \parallel \hat{p}_{\mathcal{A}} \right) = D_{KL} \left(\mathbb{E} [\text{prior}_{\mathcal{A},stu}] \parallel \mathbb{E} [\text{posterior}_{\mathcal{A},stu}] \right). \quad (10.12)$$

The right side of (10.12) is closely related to Bayesian surprise [92], which takes the form of the KL divergence from the prior to posterior. If we remove the expectations and switch the roles between the prior and posterior, we get the exact formula for Bayesian surprise. Both functionals capture the idea of comparing the statistical difference between the prior and posterior. Therefore, the efficacy of concept learning is analogous to seeking (informational) surprise in the learning process.

However, the subtlety in (10.12) where we exchange the prior and posterior, makes a distinction from Bayesian surprise due to the asymmetry of KL divergence. Adopting (10.12) as the objective function tends to produce rules about what the original data maker avoided doing, while the other way around produces what the original data maker liked to do. So, we treat it as a domain-dependent design choice. For instance, we have indeed used (10.12) in our current implementations of automatic music theorists (more details in Part III), given that music composition rules are often taught as prohibitions (e.g. “parallel fifths/octaves are bad”, “never double the tendency tones”). There are more in-depth and information-theoretic discussions on this point [93–95].

10.3 THE ELASTIC STUDENT: RULE BREAKING

Built on top of the vanilla student (Section 10.1.2), this section presents an elastic student. This more advanced student upgrades the vanilla version by being able to proactively choose a reasonable subset of given rules to realize. In general, abstraction and realization are bilateral processes that are key in deriving intelligence and creativity. In this thesis, the two processes are approached through *probabilistic rules*: high-level principles that reveal invariances within similar yet diverse examples. Under a probabilistic setting for discrete input spaces, we focus on the rule realization problem which generates input sample distri-

butions that follow the given rules. More ambitiously, we go beyond a mechanical realization that takes whatever is given, but instead ask for proactively selecting reasonable rules to realize and thus intentionally breaking the unselected rules.

This goal is demanding in practice, since the initial rule set may not always be consistent and thus intelligent compromises are needed. We formulate both rule realization and selection as two strongly connected components within a single and symmetric bi-convex problem, and derive an efficient algorithm that works at large scale.

10.3.1 Problem Formulation for the Elastic Student

Probabilistic Rule System. The interrelation between abstraction and realization $(X, p_X) \leftrightarrow (\mathcal{A}, p_{\mathcal{A}})$ can be formalized by a linear equation: $Ap = b$, where $A \in \{0, 1\}^{m \times n}$ represents a partition ($A_{ij} = 1$ if and only if x_j is assigned to the i th cell in the partition), and $p = p_X, b = p_{\mathcal{A}}$ are probability distributions of the raw input space and the high-level abstraction space, respectively. In the sequel, we re-rotate a rule by the pair (A, b) , so realizing this rule becomes solving the linear equation $Ap = b$. More interestingly, given a set of rules: $(A^{(1)}, b^{(1)}), \dots, (A^{(K)}, b^{(K)})$, the realization of all of them involves finding a p such that $A^{(r)}p = b^{(r)}$, for all $r = 1, \dots, K$. In this case, we form a *probabilistic rule system* by stacking all rules into one single linear system:

$$A = \begin{bmatrix} A^{(1)} \\ \vdots \\ A^{(K)} \end{bmatrix} \in \{0, 1\}^{m \times n}, \quad b = \begin{bmatrix} b^{(1)} \\ \vdots \\ b^{(K)} \end{bmatrix} \in [0, 1]^m. \quad (10.13)$$

We call $A_{i,:}^{(r)}p = b_i^{(r)}$ a rule *component* (i.e. a concept and its designated probability mass, essentially an information element defined by Shannon), and $m_r = \dim(b^{(r)})$ the size (the number of components or the number of concepts) of a rule.

A Unified Framework. We detail a unified framework for simultaneous rule realization and selection. Recall rules themselves can be inconsistent, e.g. rules learned from different contexts can conflict. So given an inconsistent rule system, we can only achieve $Ap \approx b$. To best realize the possibly inconsistent rule system, we solve for $p \in \Delta^n$ by minimizing the error $\|Ap - b\|_2^2 = \sum_r \|A^{(r)}p - b^{(r)}\|_2^2$, the sum of the Brier scores from every individual rule. This objective does not differentiate rules (or their components) in the rule system, which typically yields a solution that satisfies all rules approximately and achieves a small error on average. This performance, though optimal in the averaged sense, is somewhat disappointing since most often no rule is satisfied exactly (error-free). Contrarily, a human would typically

make a clear separation: follow some rules exactly and disregard others even at the cost of a larger realization error. The decision made on rule selection usually manifests the style of a person and is a higher level intelligence that we aim for. In this pursuit, we introduce a fine-grained set of weights $w \in \Delta^m$ to distinguish not only individual rules but also their components. The weights are estimates of relative importance, and are further leveraged for rule selection. This yields a weighted error, which is used to measure realization quality:

$$\mathcal{E}(p, w; A, b) = (Ap - b)^\top \text{diag}(w)(Ap - b). \quad (10.14)$$

We see that under the current setting, the first challenge concerns the curse of dimensionality for p , while the second concerns the selectivity for w . We introduce two penalty terms, one each for p and w , to tackle the two challenges, and propose the following bi-convex optimization problem as the unified framework:

$$\begin{aligned} & \text{minimize} && \mathcal{E}(p, w; A, b) + \lambda_p P_p(p) + \lambda_w P_w(w) && (10.15) \\ & \text{subject to} && p \in \Delta^n, w \in \Delta^m. \end{aligned}$$

Despite contrasting purposes, both penalty terms, $P_p(p)$ and $P_w(w)$, adopt the same high-level strategy of exploiting *group structures*¹ in p and w . Regarding the curse of dimensionality, we exploit the group structure of p by grouping p_j and $p_{j'}$ together if the j th and j' th columns of A are identical, partitioning p 's coordinates into K' groups: $g'_1, \dots, g'_{K'}$ where K' is the number of distinct columns of A . This grouping strategy uses the fact that in a simplex-constrained linear system, we cannot determine the individual p_j s within each group but only their sum. We later show the resulting group structure of p is essential in dimensionality reduction (when $K' \ll n$) and has a deeper interpretation regarding abstraction levels. Regarding the rule-level selectivity, we exploit the group structure of w by grouping weights together if they are associated with the same rule, partitioning w 's coordinates into K groups: g_1, \dots, g_K where K is the number of given rules. Based on the group structures of p and w , we introduce their corresponding group penalties as follows:

$$P_p(p) = \|p_{g'_1}\|_1^2 + \dots + \|p_{g'_{K'}}\|_1^2, \quad (10.16)$$

$$P_w(w) = \sqrt{m_1} \|w_{g_1}\|_2^1 + \dots + \sqrt{m_K} \|w_{g_K}\|_2^1. \quad (10.17)$$

One can see the symmetry here: group penalty (10.16) on p is a squared, unweighted $L_{2,1}$ -

¹When talking about the elastic student and rule breaking in this thesis, the term *group* has nothing to do with group theory, but group sparsity instead.

norm, which is designed to secure a unique solution that favors more randomness in p for the sake of diversity in data generation; group penalty (10.17) on w is a weighted $L_{1,2}$ -norm (group lasso), which enables rule selection. However, there is a pitfall of the group lasso penalty when deployed in Problem (10.15): the problem has multiple global optima that are indefinite about the number of rules to pick (e.g. selecting one rule and ten consistent rules are both optimal). To give more control over the number of selections, we finalize the penalty on w as the group elastic net that blends between a group lasso penalty and a ridge penalty:

$$P_w(w) = \alpha P'_w(w) + (1 - \alpha) \|w\|_2^2, \quad 0 \leq \alpha \leq 1, \quad (10.18)$$

where α balances between rule elimination (less rules) and selection (more rules).

Model Interpretation. Problem (10.15) is a bi-convex problem: fixing p it is convex in w ; fixing w it is convex in p . The symmetry between the two optimization variables further gives us the reciprocal interpretations of the rule realization and selection problem: given p , the realization, we can *analyze* its style by computing w ; given w , the style, we can *realize* it by computing p and further sample from it to obtain data that matches the style. The roles of the hyperparameters λ_p and (λ_w, α) are quite different. In setting λ_p sufficiently small, we secure a unique solution for the rule realization part. However, for the rule selection part, what is more interesting is that adjusting λ_w and α allows us to guide the overall data generation towards different directions, e.g. conservative (less strictly obeyed rules) versus liberal (more loosely obeyed rules).

Model Properties. We state two properties of the bi-convex problem (10.15) as the following theorems whose proofs are omitted for brevity. Both theorems involve the notion of group selective weight. We say $w \in \Delta^m$ is *group selective* if for every rule in the rule set, w either drops it or selects it entirely, i.e. either $w_{g_r} = 0$ or $w_{g_r} > 0$ element-wisely, for any $r = 1, \dots, K$. For a group selective w , we further define $\text{supp}_g(w)$ to be the selected rules, i.e. $\text{supp}_g(w) = \{r \mid w_{g_r} > 0 \text{ element-wisely}\} \subset \{1, \dots, K\}$.

Lemma 10.1. *Fix any $p \in \mathbb{R}^n$, the bi-convex problem (10.15) is reduced to a convex problem with respect to w :*

$$\begin{aligned} & \text{minimize} \quad \sum_{r=1}^K e_{g_r}^\top w_{g_r} + \lambda_w \left(\alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + (1 - \alpha) \|w\|_2^2 \right) & (10.19) \\ & \text{subject to} \quad \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1. \end{aligned}$$

Let w^* be a solution to problem (10.19), then w^* is group selective if $\lambda_w > \alpha^{-1} \|e\|_\infty$.

Proof. (Proof by contradiction.) Assume w^* is not group selective when $\lambda_m > \alpha^{-1}\|e\|_\infty$, i.e. there exists $r \in \{1, \dots, K\}$ such that

$$\|w_{g_r}^*\|_2 \neq 0, \text{ but } (w_{g_r}^*)_i = 0, \text{ for some } i. \quad (10.20)$$

Let $L(w)$ be problem (10.19)'s objective function whose partial derivative $\partial L/\partial(w_{g_k})_j$ for any group k such that $\|w_{g_k}^*\|_2 \neq 0$ is

$$\frac{\partial L}{\partial(w_{g_k})_j}(w^*) = (e_{g_k})_j + \lambda_w \left(\alpha \sqrt{m_r} \frac{(w_{g_k}^*)_j}{\|w_{g_k}^*\|_2} + 2(1 - \alpha)(w_{g_k}^*)_j \right). \quad (10.21)$$

For r, i in particular, we have

$$\frac{\partial L}{\partial(w_{g_r})_i}(w^*) = (e_{g_r})_i. \quad (10.22)$$

Within group r , let $j^* = \arg \max_j (w_{g_r}^*)_j$. Since $\sum_{j=1}^{m_r} (w_{g_r}^*)_j^2 = \|w_{g_r}^*\|_2^2 > 0$, we must have

$$(w_{g_r}^*)_{j^*} \geq \frac{1}{\sqrt{m_r}} \|w_{g_r}^*\|_2. \quad (10.23)$$

Take the direction vector $\delta^t \in \mathbb{R}^m$ such that

$$(\delta_{g_k}^t)_j = \begin{cases} t & \text{if } k = r, j = i \\ -t & \text{if } k = r, j = j^* \\ 0 & \text{otherwise,} \end{cases} \quad (10.24)$$

where $t > 0$ is sufficiently small such that $w^* + \delta^t \in \Delta^m$. However,

$$\begin{aligned} \left\langle \frac{\partial L}{\partial w^*}, \delta^t \right\rangle &= t \cdot (e_{g_r})_i - t(e_{g_r})_{j^*} - t\lambda_w \left(\alpha \sqrt{m_r} \frac{(w_{g_r}^*)_{j^*}}{\|w_{g_r}^*\|_2} + 2(1 - \alpha)(w_{g_r}^*)_{j^*} \right) \\ &\leq t \cdot (e_{g_r})_i - t\lambda_w \alpha \sqrt{m_r} \frac{(w_{g_r}^*)_{j^*}}{\|w_{g_r}^*\|_2} \end{aligned} \quad (10.25)$$

$$\leq t \cdot ((e_{g_r})_i - \lambda_w \alpha) \quad (10.26)$$

$$< 0 \quad (10.27)$$

where inequality (10.25) holds since $(e_{g_r})_{j^*} \geq 0, \alpha \leq 1, (w_{g_r}^*)_{j^*} \geq 0$; inequality (10.26) holds because of the lower bound (10.23); inequality (10.27) holds because of the condition $\lambda_w > \alpha^{-1}\|e\|_\infty$. The negative inner product in the above implies that δ^t is a descent direction,

which gives $L(w^* + \delta^t) < L(w^*)$. This contradicts the fact that w^* is a minimizer, therefore, nullifies the assumption that w^* is not group selective and completes the proof.

Theorem 10.1. Fix any $\lambda_p > 0, \alpha \in [0, 1]$. Let $(p^*(\lambda_w), w^*(\lambda_w))$ be a solution path to the bi-convex problem (10.15).

(1) $w^*(\lambda_w)$ is group selective, if $\lambda_w > 1/\alpha$.

(2) $\|w_{g_r}^*(\lambda_w)\|_2 \rightarrow \sqrt{m_r}/m$ as $\lambda_w \rightarrow \infty$, for $r = 1, \dots, K$.

Proof. For part (1), recall that $A^{(r)}p, b^{(r)} \in \Delta^{m_r}$ are both probability distributions regarding the r th rule. Hence, every element of $e_{g_r} = (A^{(r)}p - b^{(r)})^2$ is bounded in $[-1, 1]$, for any $r = 1, \dots, K$. That is, $e_i \in [-1, 1]$ for all i , or equivalently $\|e\|_\infty \leq 1$. Notice that $w^*(\lambda_w)$ is the solution to problem (10.19) for $p^*(\lambda_w)$. Then by Lemma 10.1, $w^*(\lambda_w)$ is group selective since $\lambda_w > 1/\alpha \geq \|e\|_\infty/\alpha$.

For part (2), when $\lambda_w \rightarrow \infty$, the first two terms in the objective vanish, problem (10.15) is

$$\begin{aligned} & \text{minimize} && \alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + (1 - \alpha) \|w\|_2^2 && (10.28) \\ & \text{subject to} && \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1. \end{aligned}$$

Note that the solution w^* to problem (10.28) must have uniform mass within each group:

$$(w_{g_r}^*)_i = \|w_{g_r}^*\|_1/m_r, \quad \text{for all } i = 1, \dots, m_r, \text{ and for all } r = 1, \dots, K. \quad (10.29)$$

As a consequence, the group lasso penalty is always constant:

$$\sum_{r=1}^K \sqrt{m_r} \|w_{g_r}^*\|_2 = \sum_{r=1}^K \sqrt{m_r} \frac{\|w_{g_r}^*\|_1}{\sqrt{m_r}} = \|w^*\|_1 = 1. \quad (10.30)$$

Under the uniformity condition (10.29), the ridge penalty

$$\|w\|_2^2 = \sum_r \|w_{g_r}\|_2^2 = \sum_r \frac{\|w_{g_r}\|_1^2}{m_r} \geq \frac{(\sum_r \|w_{g_r}\|_1)^2}{\sum_r m_r} = \frac{1}{m}, \quad (10.31)$$

by the Cauchy-Schwarz inequality. The equality holds when

$$\frac{\|w_{g_r}^*\|_1}{\sqrt{m_r}} = \gamma \sqrt{m_r}, \quad \text{for some constant } \gamma. \quad (10.32)$$

That is $\|w_{g_r}^*\|_1 = \gamma m_r$. Then $\sum_r \|w_{g_r}^*\|_1 = 1$ gives $\gamma = 1/m$, which further yields

$$\|w_{g_r}^*\|_2 = \frac{\|w_{g_r}^*\|_1}{\sqrt{m_r}} = \gamma \sqrt{m_r} = \frac{\sqrt{m_r}}{m}. \quad (10.33)$$

This completes the proof and further shows that w^* is actually the uniform distribution on Δ^m , since applying condition (10.29) to $\|w_{g_r}^*\|_1 = \gamma m_r = m_r/m$ gives $w_i^* = 1/m$ for all i .

Theorem 10.2. *For $\lambda_p = 0$ and any $\lambda_w > 0, \alpha \in [0, 1]$, let (p^*, w^*) be a solution to problem (10.15). We define $\mathcal{C} \subset 2^{\{1, \dots, K\}}$ such that any $C \in \mathcal{C}$ is a consistent (error-free) subset of the given rule set. If $\text{supp}_{\mathbf{g}}(w^*) \in \mathcal{C}$, then $\sum_{r \in \text{supp}_{\mathbf{g}}(w^*)} m_r = \max \{ \sum_{r \in C} m_r \mid C \in \mathcal{C} \}$.*

Proof of Theorem 2. x When $\lambda_p = 0$, problem (10.15) becomes

$$\begin{aligned} & \text{minimize} && \mathcal{E}(p, w; A, b) + \lambda_w \left(\alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + (1 - \alpha) \|w\|_2^2 \right) && (10.34) \\ & \text{subject to} && \mathbf{0} \preceq p \preceq \mathbf{1}, \quad \mathbf{1}^\top p = 1, \\ & && \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1. \end{aligned}$$

Under the error-free condition, (p^*, w^*) is also the solution to the following problem

$$\begin{aligned} & \text{minimize} && \lambda_w \alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + \lambda_w (1 - \alpha) \|w\|_2^2 && (10.35) \\ & \text{subject to} && \text{supp}_{\mathbf{g}}(w) \in \mathcal{C} \\ & && \mathcal{E}(p, w; A, b) = 0 \\ & && \mathbf{0} \preceq p \preceq \mathbf{1}, \quad \mathbf{1}^\top p = 1. \\ & && \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1. \end{aligned}$$

which can be further rewritten as the following problem by the definition of \mathcal{C} :

$$\begin{aligned} & \text{minimize} && \lambda_w \alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + \lambda_w (1 - \alpha) \|w\|_2^2 && (10.36) \\ & \text{subject to} && \text{supp}_{\mathbf{g}}(w) \in \mathcal{C} \\ & && \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1. \end{aligned}$$

The above problem is further equivalent to the following problem

$$\begin{aligned}
& \text{minimize}_{C,w} && \lambda_w \alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + \lambda_w (1 - \alpha) \|w\|_2^2 && (10.37) \\
& \text{subject to} && \text{supp}_{\mathbf{g}}(w) = C, \quad C \in \mathcal{C}, \\
& && \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1.
\end{aligned}$$

Consider a class of problems $\{Q(C) \mid C \in \mathcal{C}\}$ such that each problem $Q(C)$ is formulated:

$$\begin{aligned}
& \text{minimize}_w && \lambda_w \alpha \sum_{r=1}^K \sqrt{m_r} \|w_{g_r}\|_2 + \lambda_w (1 - \alpha) \|w\|_2^2 && (10.38) \\
& \text{subject to} && \text{supp}_{\mathbf{g}}(w) = C \\
& && \mathbf{0} \preceq w \preceq \mathbf{1}, \quad \mathbf{1}^\top w = 1.
\end{aligned}$$

The solution w^* to problem (10.37) is then the solution to problem (10.38) with the minimum objective among all the problems from $\{Q(C) \mid C \in \mathcal{C}\}$, and further the corresponding p^* is probability distribution that gives $\mathcal{E}(p^*, w^*; A, b) = 0$.

Given any $C \in \mathcal{C}$, let $(p^{*,C}, w^{*,C})$ be the solution to the corresponding problem (10.38). This problem is a reduced problem (10.28) introduced in the proof of Theorem 10.1. The only difference is that here we constrain the nonzero groups to be C . Same argument for problem (10.28) gives

$$w_{g_r}^{*,C} = \frac{1}{\sum_{r \in C} m_r} \mathbf{1}_{m_r}, \quad \text{for any } r \in C. \quad (10.39)$$

Thus, the optimal objective of $Q(C)$ for a given C is

$$\lambda_w \alpha + \lambda_w (1 - \alpha) \frac{1}{\sum_{r \in C} m_r}. \quad (10.40)$$

Finally, as w^* is the $w^{*,C}$ that achieves the minimum $Q(C)$ objective among all $C \in \mathcal{C}$,

$$\sum_{r \in \text{supp}_{\mathbf{g}}(w^*)} m_r = \max \left\{ \sum_{r \in C} m_r \mid C \in \mathcal{C} \right\}. \quad (10.41)$$

Remark 10.1. *Theorem 10.1 implies a useful range of the λ_w -solution path: if λ_w is too large, w^* will converge to a known value that always selects all the rules; if λ_w is too small, w^**

can lose the guarantee to be group selective. This further suggests the termination criteria for real applications. Theorem 10.2 considers rule selection in the consistent case, where the solution selects the largest number of rule components among all other consistent rule selections. Despite the condition $\lambda_p = 0$, in practice, this theorem suggests one way of using model for a small λ_p : if the primary interest is to select consistent rules, the model is guaranteed to pick as many rule components as possible. Yet, a more interesting application is to slightly compromise consistency to achieve better selection.

10.3.2 Alternating Solvers for the Elastic Student

It is natural to solve the bi-convex problem (10.15) by iteratively alternating the update of one optimization variable while fixing the other, yielding two alternating solvers.

The p -Solver (Rule Realization). If we fix w , the optimization problem (10.15) becomes:

$$\begin{aligned} & \text{minimize} && \mathcal{E}(p, w; A, b) + \lambda_p P_p(p) && (10.42) \\ & \text{subject to} && p \in \Delta^n. \end{aligned}$$

Making a change of variable: $q_k = \mathbf{1}^\top p_{g'_k} = \|p_{g'_k}\|_1$ for $k = 1, \dots, K'$ and letting $q = (q_1, \dots, q_{K'})$, problem (10.42) is transformed to its reduced form:

$$\begin{aligned} & \text{minimize} && \mathcal{E}(p, w; A', b) + \lambda_p \|q\|_2^2 && (10.43) \\ & \text{subject to} && q \in \Delta^{K'}, \end{aligned}$$

where A' is obtained from A by removing its column duplicates. Problem (10.43) is a convex problem with a strictly convex objective, so it has a unique solution q^* . However, the solution to the original problem (10.42) may not be unique: any p^* satisfying $q_k^* = \mathbf{1}^\top p_{g'_k}^*$ is a solution to (10.42). To favor a more random p , we can uniquely determine p^* by uniformly distributing the probability mass q_k within the group g'_k : $p_{g'_k}^* = (q_k / \dim(p_{g'_k})) \mathbf{1}$, $k = 1, \dots, K'$. We solve the problem efficiently through two proposed dimensionality reduction techniques, namely *group de-overlap* and *screening*.

Dimensionality Reduction: Group De-Overlap. Problem (10.42) is of dimension n , while its reduced form (10.43) is of dimension $K' (\leq n)$ from which we can attain dimensionality reduction. In cases where $K' \ll n$, we have a huge speed-up for the p -solver; in other cases, there is still no harm to always run the p -solve from the reduced problem (10.43). Recall that we have achieved this type of dimensionality reduction by exploiting the group structure of

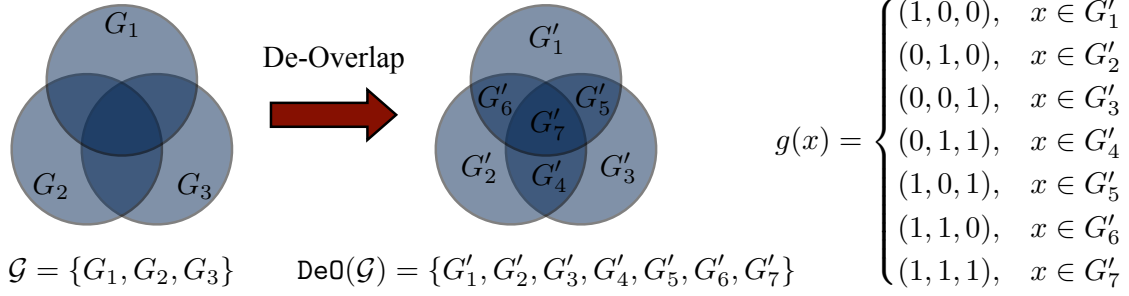


Figure 10.3: An example of group de-overlap.

p purely from a computational perspective. However, the resulting group structure has a deeper interpretation regarding abstraction levels, which is closely related to the concept of de-overlapping a family of groups, *group de-overlap* in short.

(Group De-Overlap.) Let $\mathcal{G} = \{G_1, \dots, G_m\}$ be a family of groups (a group is a non-empty set), and $G = \cup_{i=1}^m G_i$. We introduce a group assignment function $g : G \rightarrow \{0, 1\}^m$, such that for any $x \in G$, $g(x)_i = \mathbb{1}\{x \in G_i\}$, and further introduce an equivalence relation \sim on G : $x \sim x'$ if $g(x) = g(x')$. We then define the *de-overlap* of \mathcal{G} , another family of groups, by the quotient space

$$\text{DeO}(\mathcal{G}) = \{G'_1, \dots, G'_{m'}\} := G / \sim. \quad (10.44)$$

The idea of group de-overlap is simple (Figure 10.3), and $\text{DeO}(\mathcal{G})$ indeed comprises non-overlapping groups, since it is a partition of G that equals the set of equivalence classes under the equivalence relation \sim .

Given a set of rules $(A^{(1)}, b^{(1)}), \dots, (A^{(K)}, b^{(K)})$, we denote their corresponding high-level abstraction spaces by $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(K)}$, each of which is a partition of the raw input space \mathcal{X} . Let $\mathcal{G} = \cup_{k=1}^K \mathcal{A}^{(k)}$, then $\text{DeO}(\mathcal{G})$ is a new partition, thus a new high-level abstraction space, of $G = \mathcal{X}$, and is finest (may be tied) among all partitions $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(K)}$. Therefore, $\text{DeO}(\mathcal{G})$, as a summary of the rule system, delimits a lower bound on the level of abstraction produced by the given set of rules/abstractions. What coincides with $\text{DeO}(\mathcal{G})$, is the group structure of p (recall: p_j and $p_{j'}$ are grouped together if the j th and j' th columns of A are identical), since for any $x_j \in \mathcal{X}$, the j th column of A is precisely the group assignment vector $g(x_j)$. Therefore, the decomposed solve step from q^* to p^* reflects the following realization chain:

$$\{(\mathcal{A}^{(1)}, p_{\mathcal{A}^{(1)}}), \dots, (\mathcal{A}^{(K)}, p_{\mathcal{A}^{(K)}})\} \rightarrow (\text{DeO}(\mathcal{G}), q^*) \rightarrow (\mathcal{X}, p_{\mathcal{X}}), \quad (10.45)$$

where the intermediate step not only computationally achieves dimensionality reduction, but also conceptually summarizes the given set of abstractions and is further realized in the

raw input space.

Note that the σ -algebra of the probability space associated with (10.43) is precisely generated by $\text{DeO}(\mathcal{G})$. When rules are inserted into a rule system sequentially (e.g. the growing rule set from an automatic music theorist), the successive solve of (10.43) is conducted along a σ -algebra path that forms a *filtration*: nested σ -algebras that lead to finer and finer delimitations of the raw input space. In a pedagogical setting, the filtration reflects the iterative refinements of music composition from high-level principles that are taught step by step.

Dimensionality Reduction: Screening. We propose an additional technique for further dimensionality reduction when solving the reduced problem (10.43). The idea is to perform *screening*, which quickly identifies the zero components in q^* and removes them from the optimization problem. Leveraging DPC screening for non-negative lasso [96], we introduce a *screening* strategy for solving a general simplex-constrained linear least-squares problem (one can check problem (10.43) is indeed of this form):

$$\text{minimize } \|X\beta - y\|_2^2, \quad \text{subject to } \beta \succeq 0, \|\beta\|_1 = 1. \quad (10.46)$$

We start with the following non-negative lasso problem, closely related to problem (10.46):

$$\text{minimize } \phi_\lambda(\beta) := \|X\beta - y\|_2^2 + \lambda\|\beta\|_1, \quad \text{subject to } \beta \succeq 0, \quad (10.47)$$

and denote its solution by $\beta^*(\lambda)$. One can show that if $\|\beta^*(\lambda^*)\|_1 = 1$, then $\beta^*(\lambda^*)$ is a solution to problem (10.46). Our screening strategy for problem (10.46) runs the DPC screening algorithm on the non-negative lasso problem (10.47), which applies a repeated screening rule (called EDPP) to solve a solution path specified by a λ -sequence: $\lambda_{max} = \lambda_0 > \lambda_1 > \dots$. The ℓ_1 -norms along the solution path are non-decreasing: $0 = \|\beta^*(\lambda_0)\|_1 \leq \|\beta^*(\lambda_1)\|_1 \leq \dots$. We terminate the solution path at λ_t if $\|\beta^*(\lambda_t)\|_1 \geq 1$ and $\|\beta^*(\lambda_{t-1})\|_1 < 1$. Our goal is to use $\beta^*(\lambda_t)$ to predict the zero components in $\beta^*(\lambda^*)$, a solution to problem (10.46). More specifically, we assume that the zero components in $\beta^*(\lambda_t)$ are also zero in $\beta^*(\lambda^*)$, hence we can remove those components from β (also the corresponding columns of X) in problem (10.46) and reduce its dimensionality.

While in practice this assumption is true provided that we have a delicate solution path, the monotonicity of $\beta^*(\lambda)$'s support along the solution path does not hold in general [97]. Nevertheless, the assumption does hold when $\|\beta^*(\lambda_t)\|_1 \rightarrow 1$, since the solution path is continuous and piecewise linear [98]. Therefore, we carefully design a solution path in the hope of a $\beta^*(\lambda_t)$ whose ℓ_1 -norm is close to 1 (e.g. let $\lambda_i = \gamma\lambda_{i-1}$ with a large $\gamma \in (0, 1)$, while more sophisticated design is possible such as a bi-section search). To remedy the

(rare) situations where $\beta^*(\lambda_t)$ predicts some incorrect zero components in $\beta^*(\lambda^*)$, one can always leverage the KKT conditions of problem (10.46) as a final check to correct those mis-predicted components [99]. Finally, note that the screening strategy may fail when the ℓ_1 -norms along the solution path converge to a value less than 1. In these cases we can never find a desired λ_t with $\|\beta^*(\lambda_t)\|_1 \geq 1$. In theory, such failure can be avoided by a modified lasso problem which in practice does not improve efficiency much (see the supplementary material of [53]).

The w -Solver (Rule Selection). If we fix p , the optimization problem (10.15) becomes:

$$\begin{aligned} & \text{minimize} && \mathcal{E}(p, w; A, b) + \lambda_w P_w(w) \\ & \text{subject to} && w \in \Delta^m. \end{aligned} \tag{10.48}$$

We solve problem (10.48) via ADMM [100]:

$$w^{(k+1)} = \arg \min_w e^\top w + \lambda_w P_w(w) + \frac{\rho}{2} \|w - z^{(k)} + u^{(k)}\|_2^2, \tag{10.49}$$

$$z^{(k+1)} = \arg \min_z I_{\Delta^m}(z) + \frac{\rho}{2} \|w^{(k+1)} - z + u^{(k)}\|_2^2, \tag{10.50}$$

$$u^{(k+1)} = u^{(k)} + w^{(k+1)} - z^{(k+1)}. \tag{10.51}$$

In the w -update (10.49), we introduce the error vector $e = (Ap - b)^2$ (element-wise square), and obtain a closed-form solution by a soft-thresholding procedure [101]: for $r = 1, \dots, K$,

$$w_{g_r}^{(k+1)} = \left(1 - \frac{\lambda_w \alpha \sqrt{m_r}}{(\rho + 2\lambda_w(1 - \alpha)) \cdot \|\tilde{e}_{g_r}^{(k)}\|_2} \right)_+ \tilde{e}_{g_r}^{(k)}, \quad \text{where } \tilde{e}^{(k)} = \frac{\rho(z^{(k)} - u^{(k)}) - e}{\rho + 2\lambda_w(1 - \alpha)}. \tag{10.52}$$

In the z -update (10.50), we introduce the indicator function $I_{\Delta^m}(z) = 0$ if $z \in \Delta^m$ and ∞ otherwise, and recognize it as a (Euclidean) projection onto the probability simplex:

$$z^{(k+1)} = \Pi_{\Delta^m}(w^{(k+1)} + u^{(k)}), \tag{10.53}$$

which can be solved efficiently by a non-iterative method [102]. Given that ADMM enjoys a linear convergence rate in general [103] and the problem's dimension $m \ll n$, one execution of the w -solver is cheaper than that of the p -solver. Indeed, the result from the w -solver can speed up the subsequent execution of the p -solver, since we can leverage the zero components in w^* to remove the corresponding rows in A , yielding additional savings in the group de-overlap of the p -solver.

Chapter 11: Codetta: Summary and Discussions

Part II presents the algorithmic developments for abstraction construction and probabilistic rule learning, the two constituent phases of our Information Lattice Learning. Phase I can be treated as a key (deductive) preparation phase, and Phase II is the core (inductive) data-driven part of automatic concept learning. The algorithms introduced in this part are all domain-independent, but will be deployed in specific application domains in Part III to show their capability of and efficacy in conceptualizing the corresponding domains under consideration.

Chapter 12: Applicational Recapitulation (Part III)

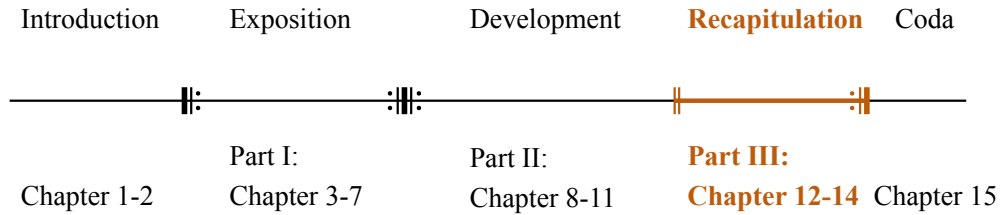


Figure 12.1: Applicational recapitulation outline.

This is the beginning of Part III. This part recapitulates Parts I and II through real-world applications. We start with music as the first application domain and thoroughly discuss automatic music concept learning. This part presents a detailed implementation of the three versions of MUS-ROVER, an automatic music theorist that distills, from sheet music, music composition rules (Chapter 13). MUS-ROVER is further wrapped into a web application and serves as an automatic music pedagogue that delivers the rules as personalized music composition lessons. To better support music ACL and music AI in general, the twin system MUS-NET is built as an online crowdsourcing platform for making and serving digital sheet music data sets (Chapter 14).

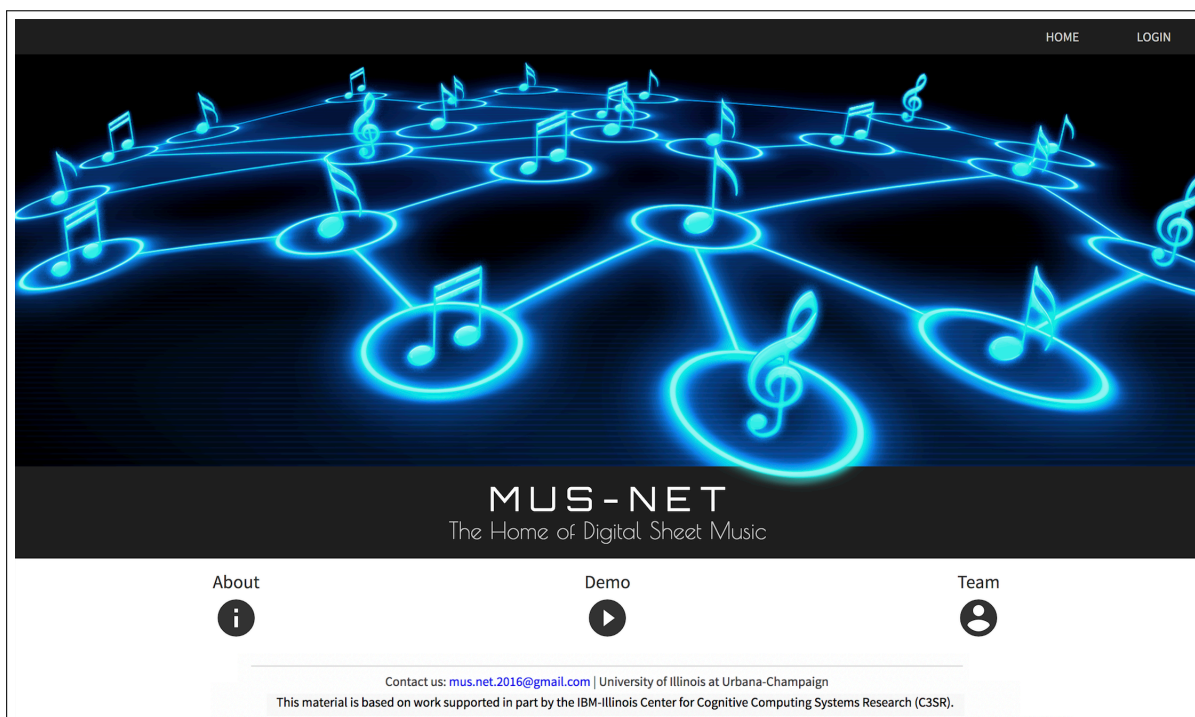
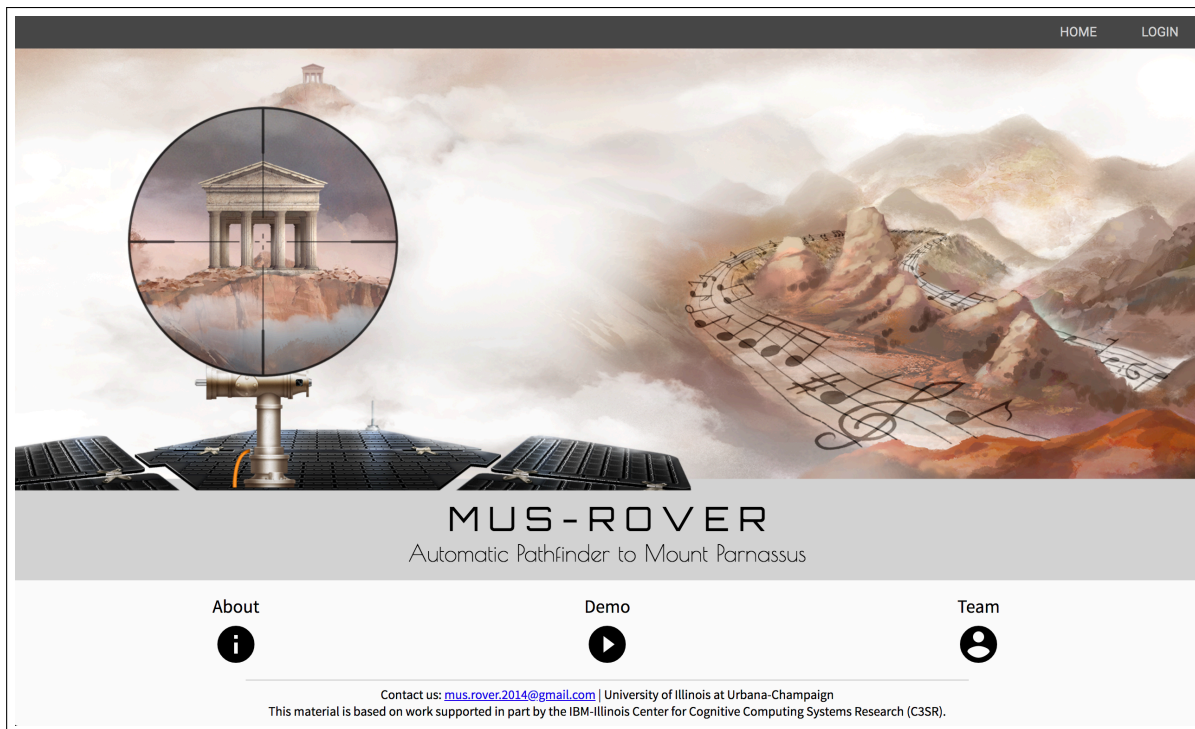


Figure 12.2: The twin projects: MUS-ROVER and MUS-NET.

Chapter 13: MUS-ROVER: An Automatic Music Theorist and Pedagogue

We present a music application of our ACL framework and ILL model to recapitulate the proposed theory and algorithms in Parts I and II. In particular, we have built an automatic music theorist and pedagogue to realize automatic music concept learning and teaching. Named MUS-ROVER, it discovers probabilistic rules of music theory from digital sheet music (the input); it further delivers the discovered rules as personalized lessons and teaches us music composition in the same style as the input music [50–52]. In this chapter, we will detail the full development path of MUS-ROVER, starting from its vanilla version MUS-ROVER I, to its successor MUS-ROVER II, as well as a special edition MUS-ROVER RB for rule breaking. Lastly, from a data set of Bach’s chorales, we compare MUS-ROVER captured rules with existing music theory to see how laws of music theory differ between a machine’s perspective and a human’s perspective.

13.1 MUSIC RAW REPRESENTATION

Given a music piece with multiple clear voices or parts, we deal with its *symbolic* representation from the sheet music rather than a waveform representation from the audio signals. In this thesis, we use J. S. Bach’s four-part (SATB) chorales as examples, and only pick the core compositional ingredients—pitches and their durations—as the raw representation of a piece, filtering other information such as meters, measures, dynamics. Hence, every choral piece is treated as multiple simultaneously emitting sequences of (discrete) pitch symbols.

To numerically encode pitches, we use integral *MIDI numbers* rather than letter names. Hence, we can perform arithmetic operations (e.g. addition/subtraction, sort/argsort, modulo) on these pitch integers. We further restrict our attention to a finite set of MIDI numbers $\Omega = \{21, 22, \dots, 108\}$, establishing a bijection to the 88 piano keys: 21/108 to the leftmost/rightmost key (A0/C8) and 60 to middle C (C4).

Following this MIDI notation for pitches, we define a *sonority* (a generic term for *chord*, but not necessarily any musically-defined triads or seventh chords), by any collection of four simultaneously sounding pitches, naturally represented by a four-dimensional vector of MIDI numbers. As a result, the *sonority space* (or the *chord space*), i.e. the space of all possible sonorities, is Ω^4 , which is a finite subset of \mathbb{Z}^4 . Here, this sonority space is the data space that we mentioned in the general ACL setting, i.e. $X = \Omega^4$; thus, a sonority distribution, either an unconditional distribution or a conditional one, is the raw data distribution that we want to derive concept from.



60	60	60	60	60	60	67	67	64	64	64	62	60	60
55	55	55	55	57	57	55	55	55	55	55	55	52	52
52	52	52	52	53	52	50	50	48	48	47	47	48	48
36	36	48	48	45	45	47	47	48	48	43	43	45	45

Figure 13.1: J. S. Bach, Aus meines Herzens Grunde in C major, mm. 1–3 (top), and its MIDI matrix representation (bottom).

We represent a four-part chorale as a four-row matrix $M \in \Omega^{4 \times N}$, whose entries are MIDI numbers. The rows represent the horizontal *melodies* in each voice: the 1st, 2nd, 3rd, and 4th row of M correspond to soprano, alto, tenor, and bass, respectively. The columns represent the vertical four-pitch sonorities, where each column has unit duration equaling the greatest common divisor (gcd) of note durations in the piece. For instance, if the choral piece is composed from quarter notes and dotted quarter notes only (the gcd of these two types of notes is an eighth note), then a quarter note C4 will result in a sequence of repeated 60s spanning 2 consecutive columns (3 columns for a dotted quarter note). The MIDI matrix is all that is needed in the subsequence computation, which is referred as a piece’s *raw* representation. Figure 13.1 gives an example of numerically transcribing a chorale excerpt into its MIDI matrix representation. In this example, every column in the MIDI matrix denotes the unit duration of an eighth note in the sheet music.

13.2 MUS-ROVER I

Throughout music history, theorists have identified rules that capture the decisions of composers. Here we ask: “Can a machine behave like a music theorist?” We present MUS-

ROVER I, a self-learning system for automatically discovering rules from symbolic music. MUS-ROVER I performs feature-induced concept learning via n -gram models to extract probabilistic music composition rules—statistical patterns on feature-induced abstractions. We evaluate MUS-ROVER I on Bach’s four-part (SATB) chorales, demonstrating that it can recover known rules, as well as identify new, characteristic patterns for further study. We discuss how the extracted rules can be used in both machine and human composition.

13.2.1 Overview

For centuries, music theorists have developed concepts and rules to describe the regularity in music compositions; music pedagogues have documented commonly agreed upon compositional rules into textbooks (e.g. *Gradus ad Parnassum*) to teach composition. With recent advances in AI, computer scientists have hardcoded these rules into programs that automatically generate different styles of music [49, 104, 105]. However, this thesis studies the *reverse* process of rule-based music compositions, and poses the question: can a machine independently extract from symbolic music data (e.g. digital sheet music), compositional rules that are instructive to both machines and humans?

This section presents MUS-ROVER I, a self-learning system for discovering compositional rules from raw music data (i.e. pitches and their durations). As described in the vanilla “teacher \rightleftharpoons student” loop (Section 10.1), the rule-learning process is implemented through an iterative loop between a *generative* model (“student”) that emulates the input’s musical style by satisfying a set of learned rules, and a *discriminative* model (“teacher”) that proposes additional rules to guide the student closer to the target style. This loop produces a rule book and a set of reading instructions customized for different types of users.

MUS-ROVER I is designed to extract rules from four-part music performed by single-line instruments. Compositional rules (output) are represented as probability distributions of feature-induced abstractions from raw music data. The rover leverages an evolving series of n -gram models over these higher-level abstraction spaces to capture potential rules from both horizontal and vertical dimensions of the music texture.

We train the rover on the C scores of Bach’s four-part (SATB) chorales, which have been an attractive corpus for analyzing knowledge of voice leading, counterpoint, and tonality due to their relative uniformity of rhythm [106, 107]. We show that MUS-ROVER I is able to automatically recover compositional rules for these chorales that have been previously identified by music theorists. In addition, we present new, human-interpretable rules discovered by MUS-ROVER that are characteristic of Bach’s chorales.

13.2.2 Literature Review

Throughout history, researchers have been building expert systems for both automatically analyzing and automatically generating music. Many analyzers leverage predefined concepts (e.g. chords, roman numerals, tonal functions) to annotate music parameters in a pedagogical process [106], or statistically measure a genre’s accordance with standard music theory [107]. Similarly, automatic song writers such as EMI [104] and GenJem [105] rely on explicit, ad-hoc coding of known rules to generate new compositions [108].

On the other hand, other systems generate music by learning statistical models such as HMMs and neural networks that capture domain knowledge, in terms of patterns, from data [109, 110]. Recent advances in deep learning take a step further, enabling knowledge discovery via feature learning directly from raw data [11, 61, 111]. However, the learned, high-level feature are implicit and non-symbolic with post-hoc interpretations, and often not directly comprehensible or evaluable.

MUS-ROVER I both automatically extracts rules from raw data—without prior encoding any domain knowledge—and ensures that the rules are interpretable by humans. Interpretable machine learning has studied systems with similar goals in other domains [112, 113].

13.2.3 Learning Model

Self-Learning Loop. MUS-ROVER I introduces a “teacher \rightleftharpoons student” model to extract compositional rules for writing 4-part chorales [50, 51]. The model is implemented by a self-learning loop between a *generative* component (student) and a *discriminative* component (teacher), where both entities cooperate to iterate through the rule-learning process (Figure 13.2). The student starts as a *tabula rasa* that picks pitches uniformly at random to form sonorities (a generic term for chord) and sonority progressions. The teacher compares the student’s writing style (represented by a probabilistic model) with the input style (represented by empirical statistics), identifying one feature per iteration that best reveals the gap between the two styles, and making it a rule for the student to update its probabilistic model. As a result, the student becomes less and less random by obeying more and more rules, and thus, approaches the input style.

Evolving n -Grams on Feature-Induced Abstractions. MUS-ROVER I employs a series of n -gram models (with *words* being vertical features) to extract horizontal rules that govern the transitions of the sonority abstractions. It also implicitly learns something about the rhythms, since the transition model probabilistically tells whether to stay or to alter. All n -grams encapsulate copies of self-learning loops to accomplish rule extractions in their

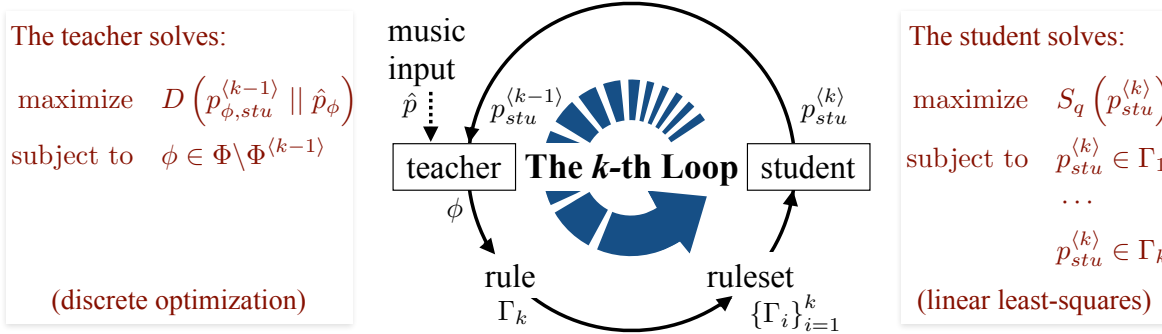


Figure 13.2: MUS-ROVER’s self-learning loop (the k th iteration). The teacher (discriminator) takes as inputs the student’s latest style $p_{stu}^{(k-1)}$ and the input style \hat{p} , and identifies a feature ϕ through which the two styles manifest the largest gap $D(\cdot||\cdot)$. The identified feature is then made into a rule (a constraint set Γ_k), and augments the ruleset $\{\Gamma_i\}_{i=1}^k$. The student (generator) takes as input the augmented ruleset to update its writing style into $p_{stu}^{(k)}$, and favors creativity, i.e. more possibilities, by maximizing the Tsallis entropy S_q subject to the rule constraints. In short, the teacher extracts rules while the student applies rules; both perform their tasks by solving optimization problems.

contexts. Starting with unigram, MUS-ROVER I gradually evolves to higher order n -grams by initializing an n -gram student from the latest $(n-1)$ -gram student. While the unigram model only captures vertical rules such as concepts of intervals and triads, the bigram model searches for rules about sonority progressions such as contrapuntal motions and resolutions. MUS-ROVER I only implements unigram and bigram models; however, higher order n -grams with wider horizontal visions can be trained similarly. The rover’s n -gram models operate on high-level feature-induced abstraction spaces, which is in stark contrast with many other n -gram applications in which the words are the raw inputs. In other words, a higher-order n -gram in the rover shows how vertical features (high-level abstractions) transition horizontally, as opposed to how a specific chord is followed by other chords (low-level details). Therefore, MUS-ROVER I does not suffer from low-level variations in the raw inputs, highlighting a greater generalizability.

13.2.4 Experiments with Bach’s Chorales

MUS-ROVER I outputs two main products: 1) a rule book on Bach’s chorales, and 2) personalized rule-learning traces for reading the book. The rule book records a lengthy list of rules that summarizes the statistics of Bach’s chorales from all possible angles (features). The rule-learning traces suggest empirical ways to read this list of conceptually entangling rules—rules that are implied from others.

Unigram Rule (ϕ, p_ϕ) Catalog			
<i>Index</i>	<i>Window</i>	<i>Descriptor</i>	<i>Entropy(p_ϕ)</i>
1	(1,4)	order	0.000
2	(1,3)	order	0.006
11	(1,2,3,4)	order	0.691
12	(1,)	pitch12	2.934
16	(1,4)	interv12	3.066
32	(2,3,4)	interv12	5.348
52	(1,2,3,4)	interv12	7.090
63	(1,2,3,4)	pitch	10.091

Table 13.1: An excerpt of the unigram rule catalog. The complete catalog includes 63 rules sorted by the Shannon entropies of their feature distributions. The windows use 1, 2, 3, 4 to denote SATB. In the Descriptor column, order is a fine-tuned `argsort` operator that handles ties in a definitive way; pitch12 is a coordinate-wise `mod12` operator; interv12 is the module version of a `diff` operator.

A Rule Book on Bach’s Chorales. MUS-ROVER I independently writes a rule book summarizing the statistical structure of Bach’s chorales. Even though the book is authored by a machine, it is designed to be readable by humans. Inspecting the rule book allows us to compare the machine-generated rules to our knowledge, and gives us concrete cases from Bach that exemplify the rules. Further, it opens the opportunity for humans to learn music theory from a machine-generated reference.

Chapter-1: Unigram Rules. The opening chapter records all the unigram rules, whose associated features are automatically generated from the feature pool. All the 63 unigram rules are sorted by the *Shannon entropies* of the feature distributions, a surrogate for human memorability [89]: rules that are more deterministic are easier for humans to memorize. Table 13.1 shows an excerpt of the (sorted) catalog of the unigram rules. The complete catalog delimits the universe of all rules that are reachable via the feature pool, which demonstrates MUS-ROVER’s exploration capacity. The first eleven rules in the catalog specify the pitch orderings between/among voices, all of which suggest that the pitch in a higher voice should sound higher. These rules, though less interesting, are consistent with our pedagogical restrictions on voice crossing. The 12th rule considers the soprano voice, and its descriptor $d_{pitch12}$ is semantically equivalent to *pitch class (p.c.)*. It shows the partition of two *p.c.* sets (Figure 13.3: top), which says that the soprano line is built on a diatonic scale. The 16th rule considers the soprano and bass, and its descriptor $d_{interv12}$ is semantically equivalent to *interval class (i.c.)*. It recovers our notion of intervalic quality: consonance and dissonance (Figure 13.3: bottom).

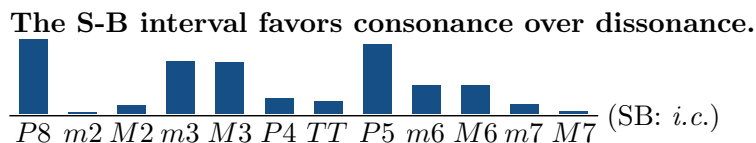
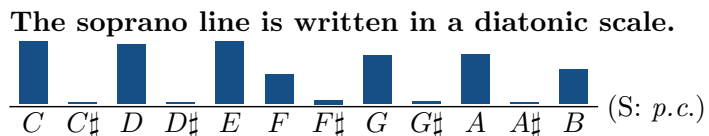


Figure 13.3: Unigram rule examples from Bach’s chorales.

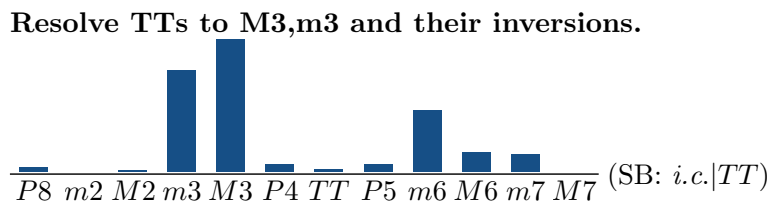
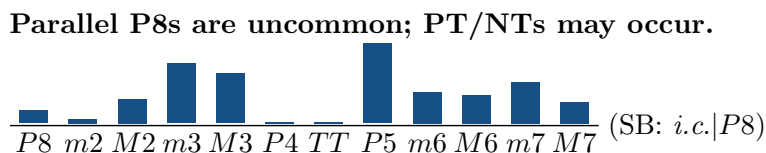
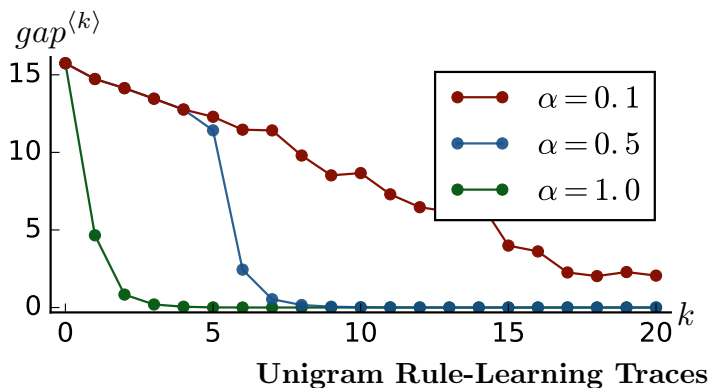


Figure 13.4: Bigram rule examples from Bach’s chorales.

Chapter-2: Bigram Rules. The second chapter records the bigram rules that are also summarized from all of the 63 features (Φ). Given a feature ϕ , the bigram rule is represented by the feature transition distribution $p_\phi(\cdot|\cdot)$. In contrast to a unigram rule where the feature distribution $p_\phi(\cdot)$ is a single pmf, the feature distribution of a bigram rule is a set of conditional pmfs $p_\phi(\cdot|\cdot) : \phi(\Omega^4) \times (\phi(\Omega^4) \cup \{*\}) \mapsto [0, 1]$ where $*$ is a specially character denoting the start symbol. In other words, we obtain a pmf by indexing a feature ϕ and the feature value of the preceding sonority $\phi(x_p)$ as the conditional. Take $\phi = d_{pitch_{12}} \circ w_{\{1,4\}}$ for example: between soprano and bass, $p_\phi(\cdot|*)$ gives the pmf of the opening *i.c.*; $p_\phi(\cdot|7)$ gives the pmf of the *i.c.* after a *P5*. Due to the large amount of conditionals for each feature, Chapter-2 is much longer than Chapter-1. Comparing the top bigram rule in Figure 13.4 with the bottom unigram rule in Figure 13.3 shows the re-distribution of the probability mass for feature $d_{pitch_{12}} \circ w_{\{1,4\}}$, the *i.c.* between soprano and bass (SB: *i.c.*). The dramatic drop of *P8* recovers the rule that avoids parallel *P8*s, while the rises of *m7*, *M7* and their inversions suggest the usage of passing/neighbor tones (PT/NTs). The bottom rule in Figure 13.4 illustrates *resolution*—an important technique used in tonal harmony—which says tritones (TTs) are most often resolved to *m3*, *M3* and their inversions. Interestingly, as a new music finding, the fifth peak (*m7*) in the pmf of this rule reveals an observation that doesn’t fall



	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 1.0$
1	(1,4), order	(1,4), order	(1,2,3), pitch
2	(1,3), order	(1,3), order	(2,3,4), pitch
3	(2,4), order	(2,4), order	(1,2,3,4), pitch12
4	(1,2), order	(1,2), order	(1,3,4), pitch
5	(2,3), order	(2,3,4), order	(1,2,4), pitch
6	(3,4), order	(1,3,4), pitch	(1,2,3,4), interv
...
E_ϵ	∞	12	6
M_ϵ	2.21	4.97	8.63

Table 13.2: Three unigram rule-learning traces ($\mathcal{T}^{(20)}$) with $\alpha = 0.1, 0.5, 1.0$. The top figure shows the footprints that mark the diminishing gaps. The bottom table records the first six rules, and shows the trade-off between efficiency and memorability ($\epsilon = 0.005$). The trace with $\alpha = 1.0$ shows the most efficiency, but the least memorability.

into the category of resolution. This transition, $TT \rightarrow m7$, is similar to the notion of escape tone (ET), which suspends the tension instead of directly resolving it. For instance, $(F4, B2) \rightarrow (F4, G2)$, which will eventually resolve to $(E4, C3)$. All of these rules are automatically identified during the rule-learning process.

Customized Rule-Learning Traces. Despite its readability for every single rule, the rule book is in general hard to read as a whole due to its length and lack of organization. The challenges are twofold: the number of rules is massive; the e-book presents rules as two unstructured lists (one for unigram, the other for bigram), whereas the rules themselves are conceptually both hierarchical and entangling. MUS-ROVER’s self-learning loop solves both challenges by offering customized *rule-learning traces*—ordered rule sequences—resulting from its iterative extraction. So, MUS-ROVER I not only outputs a rule book, but more crucially, suggests ways to read and analyze it, tailored to different types of students.

Analyzing Unigram Rules. We propose two criteria, *efficiency* and *memorability*, to

assess a rule-learning trace from the unigram model. The efficiency measures the speed in approaching Bach’s style; the memorability measures the complexity in memorizing the rules. A good trace is both efficient in imitation and easy to memorize.

To formalize these two notions, we first define a rule-learning trace $\mathcal{T}^{(k)}$ as the ordered list of the rule set $R^{(k)}$: $\mathcal{T}^{(k)} = (r^{(1)}, r^{(2)}, \dots, r^{(k)})$, and quantify the gap against Bach by the KL divergence in the raw feature space: $gap^{(k)} = D\left(\hat{p}_{\phi_{raw}|C_{bach}} \parallel p_{\phi_{raw}}^{(k)}\right)$. The *efficiency* of $\mathcal{T}^{(k)}$ with efficiency level ϵ is defined as the minimum number of iterations that is needed to achieve a student that is ϵ -close to Bach if possible:

$$E_{\epsilon}(\mathcal{T}^{(k)}) = \begin{cases} \min\{n \mid gap^{(n)} < \epsilon\}, & gap^{(k)} < \epsilon; \\ \infty, & gap^{(k)} \geq \epsilon. \end{cases} \quad (13.1)$$

The *memorability* of $\mathcal{T}^{(k)}$ is defined as the average entropy of the feature distributions from the first few efficient rules:

$$M_{\epsilon}(\mathcal{T}^{(k)}) = \frac{1}{N} \sum_{k=1}^N H(\hat{p}_{\phi^{(k)}|C_{bach}}), \quad (13.2)$$

where $N = \min\{k, E_{\epsilon}(\mathcal{T}^{(k)})\}$. We again use the average entropy to serve as a surrogate for memorability. There is a trade-off between efficiency and memorability. At one extreme, it is most efficient to just memorize $p_{\phi_{raw}}$, which takes only one step to achieve a zero gap, but is too complicated to memorize or learn. At the other extreme, it is easiest to just memorize p_{ϕ} for ordering related features, which are (nearly) deterministic but less useful, since memorizing the orderings takes you acoustically nowhere closer to Bach. There is an α parameter in the scoring function of the teacher is specially designed to balance the trade-off: a smaller α for more memorability and a larger α for more efficiency (Table 13.2).

To study the rule entangling problem, we generalize the notion of *gap* from the raw feature to all high-level features:

$$gap_{\phi}^{(k)} = D\left(\hat{p}_{\phi|C_{bach}} \parallel p_{\phi}^{(k)}\right), \quad \forall \phi \in \Phi. \quad (13.3)$$

Plotting the footprints of the diminishing gaps for a given feature reveals the (possible) implication of its associated rule from other rules. For instance, Figure 13.5 shows two sets of footprints for $\phi^{(6)}$ and $\phi^{(11)}$. By starring the iteration when the rule of interest is actually learned, we can see that $r^{(6)}$ cannot be implied from the previous rules, since learning this rule dramatically closes the gap; on the contrary, $r^{(11)}$ can be implied from the starting seven or eight rules, since the gap already drops to almost zero before learning this rule.

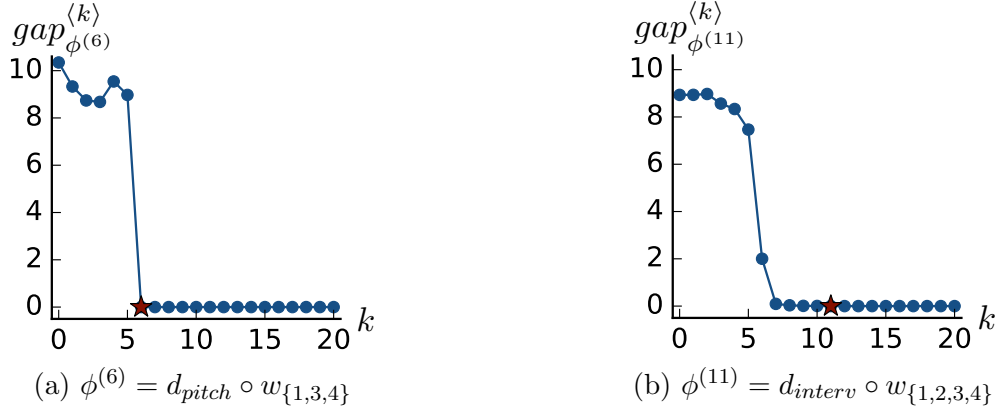


Figure 13.5: Rule entanglement: two sets of footprints that mark the diminishing gaps, both of which are from the rule-learning trace with $\alpha = 0.5$. The location of the star shows whether the associated rule is entangled (right) or not (left).

Analyzing Bigram Rules. Given a rule-learning trace in the bigram setting, the analysis on efficiency and memorability, as well as feature entanglement, remains the same. However, every trace from the bigram model is generated as a continuation of the unigram learning: the bigram student is initialized from the latest unigram student. This implies that the bigram rule set is initialized from the unigram rule set (incremental learning), rather than from an empty set. MUS-ROVER I uses the extracted bigram rules to overwrite their unigram counterparts—rules with the same features—highlighting the differences between the two language models. The comparison between a bigram rule and its unigram counterpart is key in recovering rules that are otherwise unnoticeable from the bigram rule alone, such as “Parallel P8s are avoided!” Therefore, MUS-ROVER I emphasizes the necessity of tracking a series of evolving n -grams, rather than learning from the highest possible order only. Table 13.3 exemplifies two rule-learning traces: one conditioned on the start symbol $*$; the other conditioned on a specific C major triad (76, 72, 67, 60). The A-tag denotes an addition of a new rule feature into the rule set; the U-tag denotes an update of an existing rule feature from its unigram pmf to its bigram pmf (context overwrites fundamental).

Bigram Rule-Learning Traces ($\alpha = 0.5$)		
	$x_p = *$	$x_p = (76, 72, 67, 60)$
1	(1,3), order / U	(2), pitch / A
2	(2,4), order / U	(1,3,4), pitch / U
...
9	(1,4), interv12 / A	(1,2,4), pitch / U

Table 13.3: Two bigram rule-learning traces (excerpt).

13.3 MUS-ROVER II

As pointed out earlier, music theory studies the regularity of patterns in music to capture concepts underlying music styles and composers' decisions. This section continues the study of building *automatic theorists* (rovers) to learn and represent music concepts that lead to human interpretable knowledge and further lead to materials for educating people. MUS-ROVER I took a first step in algorithmic concept learning of tonal music, studying high-level representations (concepts) of symbolic music (digital sheet music) and extracting interpretable rules for music composition. This section further studies the representation *hierarchy* through the learning process, and supports *adaptive* 2D memory selection in the resulting language model. This leads to a deeper-level interpretability that expands from individual rules to a dynamic system of rules, making the entire rule-learning process more cognitive. The outcome is a new rover, MUS-ROVER II, trained on the same set of Bach's chorales, which outputs customizable syllabi for learning compositional rules. We show comparable results to our music pedagogy, while also presenting the differences and variations.

13.3.1 Overview

Forming hierarchical concepts from low-level observations is key to knowledge discovery. In the field of artificial neural networks, deep architectures are employed for machine learning tasks, with the awareness that hierarchical representations are important [11]. Rapid progress in deep learning has shown that mapping and representing topical domains through increasingly abstract layers of feature representation is extremely effective. Unfortunately, this layered representation is difficult to interpret or use for teaching people. Consequently, deep learning models are widely used as algorithmic task performers (e.g. AlphaGo), but few act as theorists or pedagogues. In contrast, our goal is to achieve a deeper-level interpretability that explains not just what has been learned (the end results), but also what is being learned at every single stage (the process).

On the other hand, music theory studies underlying patterns beneath the music surface. It *objectively* reveals higher-level invariances that are hidden from the low-level variations. In practice, the development of music theory is an *empirical* process. Through manual inspection of large corpora of music works, theorists have summarized compositional rules and guidelines (e.g. J. J. Fux, author of *Gradus ad Parnassum*, the most influential book on Renaissance polyphony), and have devised multi-level analytical methods (e.g. H. Schenker, inventor of Schenkerian analysis) to emphasize the hierarchical structure of music, both of which have become the standard materials taught in today's music theory classes. The

objective and *empirical* nature of music theory suggests the possibility of a more advanced *automatic theorist*—statistical techniques that perform hierarchical concept learning—while its *pedagogical* purpose requires interpretability throughout the entire learning process.

The book title *Gradus ad Parnassum*, means “the path towards Mount Parnassus,” the home of poetry, music, and learning. This section presents MUS-ROVER II, an upgraded extension of MUS-ROVER I, to independently retake the path towards Parnassus. The rover acts more as a pathfinder than a generative model (e.g. LSTM), emphasizing the *path* more than the *destination*. We compare the paths taken by this improved automatic theorist to paths taken by human theorists (say Fux), studying similarities as well as pros and cons of each. Therefore, advantages from both can be jointly taken to maximize the utility in music education and research. In this section in particular, we highlight the concept hierarchy that one would not get from MUS-ROVER I, as well as enhanced syllabus personalization that one would not typically get from traditional pedagogy.

What is Inherited from MUS-ROVER I? MUS-ROVER II targets the same goal of learning interpretable music concepts. It inherits the self-learning loop (Figure 13.2), as well as the following design choices.

- (Data set and Data Representation.) We use the same data set that comprises 370 C scores of Bach’s four-part chorales. Only pitches and their durations are included in a piece’s *raw* representation, notated as a MIDI matrix whose elements are MIDI numbers for pitches. The matrix preserves the two-dimensional chorale texture, with rows corresponding to melodies, and columns to harmonies.
- (Rule Representation.) We use the same representation for high-level concepts in terms of probabilistic rules, which are unrelated to rules in propositional logic. In light of a feature-induced abstraction, a probabilistic rule is represented more directly by a feature and its probability distribution: $r = (\phi, p_\phi)$, which describes the likelihoods of the feature values. Rule satisfaction can also be translated to a linear equality constraint ($A_\phi p_{stu} = p_\phi$) in the student’s optimization problem (Γ ’s in Figure 13.2).
- (Student’s Probabilistic Model.) We still use n -gram models to represent the student’s probabilistic model, with *words* being sonority features, and keep the student’s optimization problem as it was. To reiterate the distinctions to many traditional music n -grams, we *never* run n -grams in the raw input space, but *only* collectively in the high-level feature spaces to prevent overfitting. So, rules are expressed as probabilistic laws that describe either (vertical) sonority features or their (horizontal) progressions.

What is New in MUS-ROVER II? We study *hierarchies* on features, so rules are later presented *not* just as a linear list, but as hierarchical families and sub-families (or partial information lattices). In particular, we introduce *conceptual hierarchy* that is pre-determined by feature maps, and infer *informational hierarchy* that is post-implied from an information-theoretic perspective. We upgrade the self-learning loop to *adaptively* select memories in a multi-feature multi- n -gram language model. This is realized by constructing *hierarchical filters* to filter out conceptual duplicates and informational implications. By further following the information scent spilled by *Bayesian surprise* [92], the rover can effectively localize the desired features in the feature pool.

13.3.2 Literature Review

Adversarial or Collaborative? For both MUS-ROVER I and II, the self-learning loop between the teacher (a discriminator) and student (a generator) shares great *structural* similarity to generative adversarial nets [27] and their derivatives [114, 115]. However, the working mode between the discriminator and generator is different. In current GAN-based algorithms, the adversarial components are black-boxes to each other, since both are different neural networks that are coupled only end to end. The learned intermediate representation from one model, no matter how expressive or interpretable, is *not* directly shared with the other. Contrarily, in MUS-ROVER, both models are transparent to each other (also to us): the student directly leverages the rules from the teacher to update its probabilistic model. In this sense, the learning pair in MUS-ROVER is more *collaborative* rather than *adversarial*. Consequently, not only the learned concepts have interpretations individually, but the entire learning trace is an interpretable, cognitive process.

Furthermore, MUS-ROVER and GAN contrast in the *goal* of learning and the resulting *evaluations*. The rover is neither a classifier nor a density estimator, but rather a pure representation learner that outputs high-level concepts and their hierarchies. Training this type of learner in general is challenging due to the lack of a clear objective or target [11], which drives people to consider some end task like classification and use performance on the task to *indirectly* assess the learned representations. In MUS-ROVER, we introduce information-theoretic criteria to guide the training of the automatic theorist, and in the context of music concept learning, we *directly* evaluate machine generated rules and hierarchies by comparison to those in existing music theory.

Interpretable Feature Learning. In the neural network community, much has been done to first recover disentangled representations, and then post-hoc interpret the semantics of the learned features. This line of work includes denoising autoencoders [116] and restricted

Boltzmann machines [117, 118], ladder network algorithms [119], as well as more recent GAN models [120]. In particular, InfoGAN also introduces information-theoretic criteria to augment the standard GAN cost function, and to some extent achieves interpretability for both discrete and continuous latent factors [121]. However, beyond the end results, the overall learning process of these neural networks are still far away from human-level concept learning [122], so not directly instructional to people.

Automatic Musicians. Music *theory* and *composition* form a reciprocal pair, often realized as the complementary cycle of *reduction* and *elaboration* [32] as walks up and down the multi-level music hierarchy. Accordingly, various models have been introduced to automate this up/down walk, including music generation [104, 105, 109], analysis [106], or theory evaluation [107]. In terms of methodologies, we have rule-based systems [123], language models [109, 124], and information-theoretic approaches [125, 126]. However, all of these models leverage domain knowledge (e.g. human-defined chord types, functions, rules) as part of the model inputs. MUS-ROVER takes as input *only* the raw notations (pitches and durations), and outputs concepts that are comparable to (but also different from) our domain knowledge.

13.3.3 Learning Model: Hierarchical Rule Learning

MUS-ROVER II emphasizes *hierarchy* induction in learning music representations, and divides the induction process into two stages. In Stage 1, we impose *conceptual hierarchy* as *pre-defined* structures among candidate features before the self-learning loop. In Stage 2, we infer *informational hierarchy* as *post-implied* structures through the rule learning loops.

Interpretable Features. Recall that a *feature* is a function that computes a distributed representation of the building blocks that constitute data samples. For Bach’s four-part chorales, we model every piece (a four-row matrix) as a sequence of sonorities (columns). Thus, every sonority is the building block of its composing piece (like a word in a sentence). Then a feature maps a sonority onto some feature space, summarizing an attribute. More specifically, let $\Omega = \{\mathbf{R}, \mathbf{p}_1, \dots, \mathbf{p}_n\}$ be an alphabet that comprises a rest symbol \mathbf{R} , and n pitch symbols \mathbf{p}_i . In addition, the alphabet symbols—analogue to image pixels—are manipulable by arithmetic operations, such as plus/minus, modulo, and sort. More precisely, every \mathbf{p}_i is an integer-valued MIDI number (60 for middle C, granularity 1 for semi-tone), and \mathbf{R} is a special character which behaves like a python `nan` variable. The four coordinates of every sonority $p \in \Omega^4$ denote soprano, alto, tenor, and bass, respectively. We define a *feature* as a surjective function $\phi : \Omega^4 \rightarrow \phi(\Omega^4)$, and the corresponding *feature space* by its image. As a first and brutal categorization, we say a feature (space) is *raw* (or *lowest-level*) if

$|\phi(\Omega^4)| = |\Omega^4|$, and *high-level* if $|\phi(\Omega^4)| < |\Omega^4|$. For instance, Ω^4 or any permutation of Ω^4 is a raw feature space.

MUS-ROVER II adopts the systematic way of generating the universe of interpretable features (Section 9.1). A sonority feature is constructed as the composition of a window and a descriptor. A *window* is a function that selects parts of the input sonority: $w_I : \Omega^4 \rightarrow \Omega^{|I|}$, where I is an index set. For instance, $w_{\{1,4\}}(p) = (p_1, p_4)$ selects soprano and bass. A *descriptor* is constructed inductively from a set of basis descriptors B , consisting of atomic arithmetic operations. We currently set $B = \{\text{order}, \text{diff}, \text{sort}, \text{mod}_{12}\}$, where

$$\text{diff}(x) = (x_2 - x_1, x_3 - x_2, \dots), \quad \forall x \in \Omega^2 \cup \Omega^3 \cup \Omega^4; \quad (13.4)$$

$$\text{sort}(x) = (x_{(1)}, x_{(2)}, \dots), \quad \forall x \in \Omega^2 \cup \Omega^3 \cup \Omega^4; \quad (13.5)$$

$$\text{mod}_{12}(x) = (\text{mod}(x_1, 12), \text{mod}(x_2, 12), \dots), \quad \forall x \in \Omega \cup \Omega^2 \cup \Omega^3 \cup \Omega^4; \quad (13.6)$$

and $\text{order}(x)$, a fine-tuned version of argsort , maps $x \in \Omega^2 \cup \Omega^3 \cup \Omega^4$ to a string that specifies the ordering of its elements, e.g. $\text{order}((60, 55, 52, 52)) = "4=3<2<1"$. The numbers in an order string denote the coordinates of the input vector x . Recall that we define a descriptor of length k as the composition of k bases: $d_{(k)} = b_k \circ \dots \circ b_1$, for all $b_i \in B$, where $d_{(0)}$ is the identity function. We collect the family of all possible windows: $W = \{w_I \mid I \in 2^{\{1,2,3,4\}} \setminus \{\emptyset\}\}$, and the family of all descriptors of length less than or equal to k : $D^{[k]} = \{d_{(k')} \mid 0 \leq k' \leq k\}$, and form the *feature pool*: $\Phi = \Phi^{[k+1]} := \{d \circ w \mid w \in W, d \in D^{[k]}\}$.

Feature-Induced Partition. On the one hand, a feature function has all the mathematic specifications to name the corresponding feature and feature values. On the other hand, we *only* care about the abstraction, i.e. the partition, of the input domain (Ω^4) induced by the feature but *not* the (superficial) naming of the clusters. In other words, we only identify the sonority clusters whose members are mapped to the same feature value, but not the feature value itself. A feature pool Φ induces its corresponding partition family \mathcal{P}_Φ . For two partitions $\mathcal{P}, \mathcal{Q} \in \mathcal{P}_\Phi$, we say \mathcal{P} is *finer* than \mathcal{Q} (or \mathcal{Q} is *coarser*), written as $\mathcal{P} \succeq \mathcal{Q}$, if for all $p, p' \in \Omega^4$, p, p' are in the same cluster under $\mathcal{P} \Rightarrow p, p'$ are in the same cluster under \mathcal{Q} . We say \mathcal{P} is *strictly finer*, written as $\mathcal{P} \succ \mathcal{Q}$, if $\mathcal{P} \succeq \mathcal{Q}$ and $\mathcal{Q} \not\preceq \mathcal{P}$.

Conceptual Hierarchy. Based on the binary relation “strictly finer” \succ , we construct the *conceptual hierarchy* for the partition family \mathcal{P}_Φ , and represent it as a directed acyclic graph (DAG) with nodes being partitions. For any pair of nodes v, v' , $v \rightarrow v'$ if and only if the partition referred by v is (strictly) finer than that referred by v' . The DAG grows from a single source node, which represents the finest partition—every point in the domain by itself is a cluster—and extends via the edges to coarser and coarser partitions. In terms of features, we say a feature ϕ' is at a *higher level* than another feature ϕ , if the induced

partitions satisfy $\mathcal{P}_\phi \succ \mathcal{P}_{\phi'}$. In other words, a higher-level feature induces a coarser partition that ignores lower-level details by merging clusters.

We emphasize the necessity of this multi-step process: features \rightarrow partitions \rightarrow hierarchy (DAG), rather than a simple hierarchical clustering (tree). The latter tends to lose many inter-connections due to the tree structure and its greedy manner, and more importantly, the interpretability of the partitions.

Informational Hierarchy. We infer *informational hierarchy* from a many-to-one relation, called *implication*, along a rule trace. More formally, let $\{r_i\}_{i=1}^k := \{(\phi_i, \hat{p}_{\phi_i})\}_{i=1}^k$ be the extracted trace of rules (in terms of feature and feature distribution) by the k th iteration of the self-learning loop. We say a feature ϕ is *informationally implied* from the trace $\{r_i\}_{i=1}^k$ with tolerance $\gamma > 0$, if

$$\text{gap} \left(p_{\phi,stu}^{(k)} \parallel \hat{p}_\phi \right) := D \left(p_{\phi,stu}^{(k)} \parallel \hat{p}_\phi \right) < \gamma, \quad \text{and} \quad \text{gap} \left(p_{\phi,stu}^{(k')} \parallel \hat{p}_\phi \right) \geq \gamma, \forall k' < k, \quad (13.7)$$

where $D(\cdot \parallel \cdot)$ is the KL divergence used to characterize the gap of the student’s style (probabilistic model) against Bach’s style (input). One trivial case happens when ϕ is extracted as the k th rule, i.e. $\phi = \phi_k$, then $\text{gap}(p_{\phi,stu}^{(k)} \parallel \hat{p}_{\phi'}) = 0 < \gamma, \forall \phi' \in \{\phi' \mid \mathcal{P}_\phi \succ \mathcal{P}_{\phi'}\}$, meaning that feature ϕ , once learned as a rule, informationally implies itself and all its descendants in the conceptual hierarchy. However, what is more interesting is the informational implication from other rules outside the conceptual hierarchy, which is hard for humans to “eyeball”.

Hierarchical Filters. We build *hierarchical filters* from both conceptual and informational hierarchies, for the purpose of pruning hierarchically entangled features and speeding up feature selection. This upgrades MUS-ROVER II into a more efficient, robust, and cognitive theorist. Recall the skeleton of the teacher’s optimization problem in Figure 13.2, we flesh it out as follows (also cf. the general case (10.9)):

$$\begin{aligned} & \underset{\phi \in \Phi}{\text{maximize}} && \text{gap} \left(p_{\phi,stu}^{(k-1)} \parallel \hat{p}_\phi \right) && (13.8) \\ & \text{subject to} && H(\hat{p}_\phi) \leq \delta && \text{(regularity condition)} \\ & && \phi \notin \mathfrak{C}^{(k-1)} := \{ \phi \mid \mathcal{P}_\phi \preceq \mathcal{P}_{\phi'}, \phi' \in \Phi^{(k-1)} \} && \text{(conceptual-hierarchy filter)} \\ & && \phi \notin \mathfrak{I}^{(k-1)} := \left\{ \phi \mid \text{gap} \left(p_{\phi,stu}^{(k-1)} \parallel \hat{p}_\phi \right) < \gamma \right\} && \text{(informational-hierarchy filter)} \end{aligned}$$

In the above optimization problem, Φ is the feature pool and $\phi \in \Phi$ is the optimization variable whose optimal value is used to form the k th rule: $\phi_k = \phi^*, r_k = (\phi^*, \hat{p}_{\phi^*})$. The first constraint requires the Shannon entropy of the feature distribution to be no larger than a given threshold [89]. The second constraint encodes the filter from conceptual hierarchy,

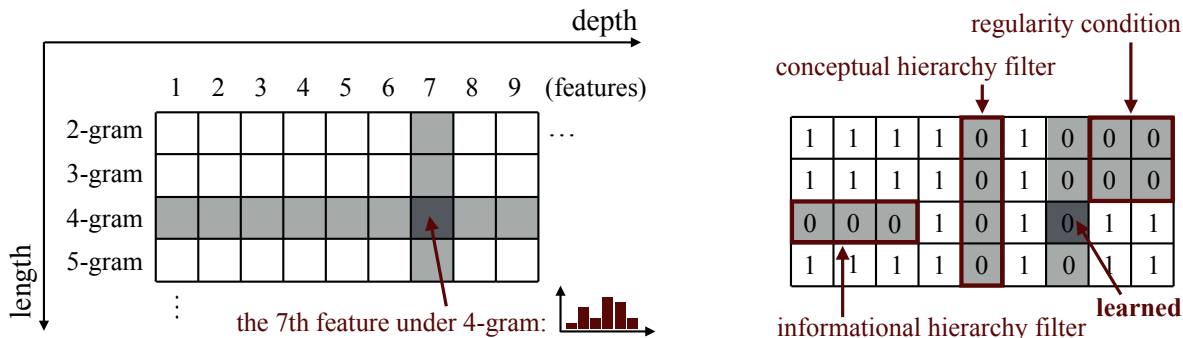


Figure 13.6: MUS-ROVER II’s two-dimensional memory (left): the length axis enumerates n -gram orders; the depth axis enumerates features; and every cell is a feature distribution. Memory mask (right): 0 marks the removal of the corresponding cell from feature selection, which is caused by a hierarchical filter or the regularity condition or (contradictory) duplication.

which prunes coarser partitions of the learned features $\Phi^{(k-1)} := \{\phi_1, \dots, \phi_{k-1}\}$. (Learning a rule is effectively learning its sub-family, since the distributions of all its descendants can be inferred exactly from the rule.) The third constraint encodes the filter from informational hierarchy, which prunes informationally implied features. There are two hyper-parameters δ and γ in the optimization problem (13.8). We often pre-select γ before the loop to express a user’s *satisfaction level*: a smaller γ signifies a meticulous user who is harder to satisfy; the threshold δ upper bounds the *entropic difficulty* of the rules, and is adaptively adjusted through the loop: it starts from a small value (easy rules first), and auto-increases whenever the feasible set of (13.8) is empty (gradually increases the difficulty when mastering the current level).

13.3.4 Learning Model: Adaptive Memory Selection

MUS-ROVER II considers a continuous range of higher order n -grams (variable memory), and adaptively picks the optimal n based on a balance among multiple criteria. The fact that every n -gram is also on multiple high-level feature spaces allows long-term memories without exhausting machine memory, while effectively avoiding overfitting.

Two-Dimensional Memory. For a range of multiple n -grams, say $n \in N = \{2, 3, \dots\}$, the feature pool is a two-dimensional memory ($N \times \Phi$)—*length* versus *depth*—for the language model (Figure 13.6: left). The length axis enumerates n -gram orders, with a longer memory corresponding to a larger n ; the depth axis enumerates features, with a deeper memory corresponding to a higher-level feature-induced abstraction. Every cell in the memory is indexed by two coordinates (n, ϕ) , referring to the feature ϕ under the n -gram, and stores

the corresponding feature distribution. As a consequence, the rule extraction task involves picking the right feature under the right n -gram, which extends the space of the optimization problem (13.8) from Φ to $N \times \Phi$. Accordingly, the constraints of (13.8) jointly forge a mask on top of the 2D memory (Figure 13.6: right).

13.3.5 Experiments with Bach’s Chorales

MUS-ROVER II’s main use case is to produce personalized syllabi that are roadmaps to learning the input style (customized paths to Mount Parnassus). By substituting the student module, users can join the learning cycle, in which they make hands-on compositions and get iterative feedback from the teacher. Alternatively, for faster experimentation, users make the student their learning puppet, which is personalized by its external parameters. Here, we discuss the latter case in detail.

Pace Control and Syllabus Customization. We present a simple yet flexible pace control panel to the users of MUS-ROVER II, enabling personalized set-up of their learning puppet. The control panel exposes four knobs: the lower bound, upper bound, and stride of the rule’s entropic difficulty ($\delta_{min}, \delta_{max}, \delta_{stride}$), as well as the satisfactory gap (γ). These four hyper-parameters together allow the user to personalize the pace and capacity of her learning experience. The entropic difficulty δ caps the Shannon entropy of a rule’s feature distribution in (13.8), a surrogate for the complexity (or memorability) of the rule [89]. It is discretized into a progression staircase from δ_{min} up to δ_{max} , with incremental δ_{stride} . The resulting syllabus starts with $\delta = \delta_{min}$, the entry level difficulty; and ends whenever $\delta \geq \delta_{max}$, the maximum difficulty that the user can handle. Anywhere in between, the loop deactivates all rules whose difficulties are beyond current δ , and moves onto the next difficulty level $\delta + \delta_{stride}$ if the student’s probabilistic model is γ -close to the input under all currently active rule features.

To showcase syllabus customization, we introduce an ambitious user who demands a faster pace and a patient user who prefers a slower one. In practice, one can collectively tune the stride parameter δ_{stride} and the gap parameter γ , with a faster pace corresponding to a larger δ_{stride} (let’s jump directly to the junior year from freshman) and a larger γ (having an A- is good enough to move onto the next level, why bother having A+). Here we simply fix δ_{stride} , and let γ control the pace. We illustrate two syllabi in Table 13.4, which compares the first ten (1-gram) rules in a faster ($\gamma = 0.5$) syllabus and a slower one ($\gamma = 0.1$). Notice the faster syllabus gives the fundamentals that a music student will typically learn in her first-year music theory class, including rules on voice crossing, pitch class set (scale), intervals, and so on (triads and seventh chords will appear later). It effectively skips the nitty-gritty

Rule Trace	Faster Pace ($\gamma = 0.5$)	Slower Pace ($\gamma = 0.1$)
1	<code>order</code> \circ $w_{\{1,2,3,4\}}$	<code>order</code> \circ $w_{\{1,2,3,4\}}$
2	<code>mod</code> ₁₂ \circ $w_{\{1\}}$	<code>order</code> \circ <code>diff</code> \circ <code>sort</code> \circ $w_{\{1,2,4\}}$ †
3	<code>mod</code> ₁₂ \circ <code>diff</code> \circ $w_{\{2,3\}}$	<code>order</code> \circ <code>diff</code> \circ <code>mod</code> ₁₂ \circ $w_{\{1,2,3\}}$ †
4	<code>mod</code> ₁₂ \circ <code>diff</code> \circ $w_{\{3,4\}}$	<code>order</code> \circ <code>diff</code> \circ <code>diff</code> \circ $w_{\{1,2,3,4\}}$ †
5	<code>diff</code> \circ <code>sort</code> \circ $w_{\{2,3\}}$	<code>order</code> \circ <code>sort</code> \circ <code>mod</code> ₁₂ \circ $w_{\{2,3,4\}}$ †
6	<code>mod</code> ₁₂ \circ $w_{\{3\}}$	<code>order</code> \circ <code>sort</code> \circ <code>mod</code> ₁₂ \circ $w_{\{1,3,4\}}$ †
7	<code>mod</code> ₁₂ \circ <code>diff</code> \circ $w_{\{1,2\}}$	<code>order</code> \circ <code>sort</code> \circ <code>mod</code> ₁₂ \circ $w_{\{1,2,3,4\}}$ †
8	<code>mod</code> ₁₂ \circ <code>diff</code> \circ $w_{\{2,4\}}$	<code>mod</code> ₁₂ \circ $w_{\{1\}}$
9	<code>diff</code> \circ $w_{\{1,2\}}$	<code>mod</code> ₁₂ \circ <code>diff</code> \circ $w_{\{2,3\}}$
10	<code>diff</code> \circ <code>sort</code> \circ $w_{\{1,3\}}$	<code>mod</code> ₁₂ \circ <code>diff</code> \circ $w_{\{3,4\}}$

Table 13.4: Customizing a syllabus († signifies rules that are skipped in the faster pace)

rules (marked by a dagger) that are learned in the slower setting. Most of these skipped rules do not have direct counterparts in music theory (such as taking the `diff` operator twice) and are not important, although occasionally the faster syllabus will skip some rules worth mentioning (such as the second rule in the slower pace, which talks about spacing among soprano, alto, and bass). Setting an appropriate pace for a user is important: a pace that is too fast will miss the whole point of knowledge discovery (jump to the low-level details too fast); a pace that is too slow will bury the important points among unimportant ones (hence, lose the big picture). The control panel is there to help a user to find the proper pace, and it can be adjusted anytime along the loop.

Fundamentals: Hierarchical 1-gram. Similar to our teaching of music theory, MUS-ROVER II’s proposed syllabus divides into two stages: fundamentals and part writing. The former is under the 1-gram setting, involving knowledge independent of the context; the latter provides online tutoring under multi- n -grams. We begin our experiments with fundamentals, and use them to illustrate the two types of feature hierarchies.

Let’s take a closer look at the two syllabi in Table 13.4. The specifications (left) and hierarchies (right) of the four common rules are illustrated in Table 13.5. The rules’ translations are below the corresponding bar charts, all of which are consistent with our music theory. Extracted from the conceptual hierarchy, the right column lists the partition sub-family sourced at each rule, which is pictorially simplified as a tree by hiding implied edges from its corresponding DAG. Every coarser partition in a sub-family is indeed a higher-level representation, but has not accumulated sufficient significance to make itself a rule. A partition will never be learned if one of its finer ancestors has been made a rule. Observe that all of the coarser partitions are not typically taught in theory classes.

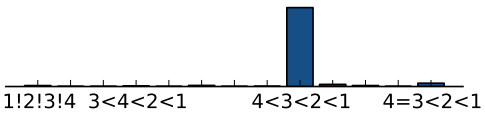
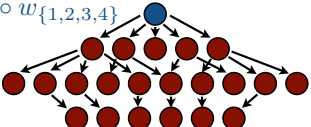
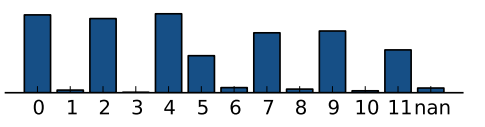
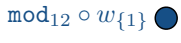
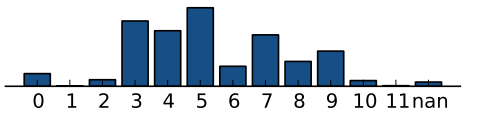

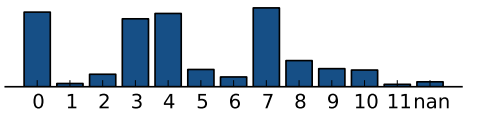
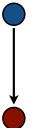
 <p>Interpretable rule (Spacing): Almost always, the soprano pitch is above the alto, alto above tenor, and tenor above bass.</p>	<p>$\text{order} \circ w_{\{1,2,3,4\}}$</p>  <p>This partition sub-family includes 21 coarser partitions, which are local orderings that are already captured by the global ordering.</p>
 <p>Interpretable rule (Scale): The soprano voice is drawn from a diatonic scale with high probability.</p>	<p>$\text{mod}_{12} \circ w_{\{1\}}$</p>  <p>This partition sub-family does not contain any other coarser partitions.</p>
 <p>Interpretable rule (Interval): The interval of the inner voices are mostly consonant (3,4,5,7,8,9), but perfect octave/unison (0) is rare due to the tight spacing between alto and tenor.</p>	<p>$\text{mod}_{12} \circ \text{diff} \circ w_{\{2,3\}}$</p>  <p>This partition sub-family contains only one coarser partition: $\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{\{2,3\}}$.</p>
 <p>Interpretable rule (Interval): The interval of the lower voices are mostly consonant, and emerges more perfect octaves due to the wide spacing between tenor and bass. Also, perfect fourth (5) is now considered as a dissonance against the bass.</p>	<p>$\text{mod}_{12} \circ \text{diff} \circ w_{\{3,4\}}$</p>  <p>This partition sub-family contains only one coarser partition: $\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{\{3,4\}}$.</p>

Table 13.5: Sample 1-gram rules and their hierarchies. The blue bubbles and the red bubbles in the right column represent (conceptually) implying and implied rules, respectively.

MUS-ROVER II measures the student’s progress from many different angles in terms of features. With respect to a feature, the gap between the student and Bach is iteratively

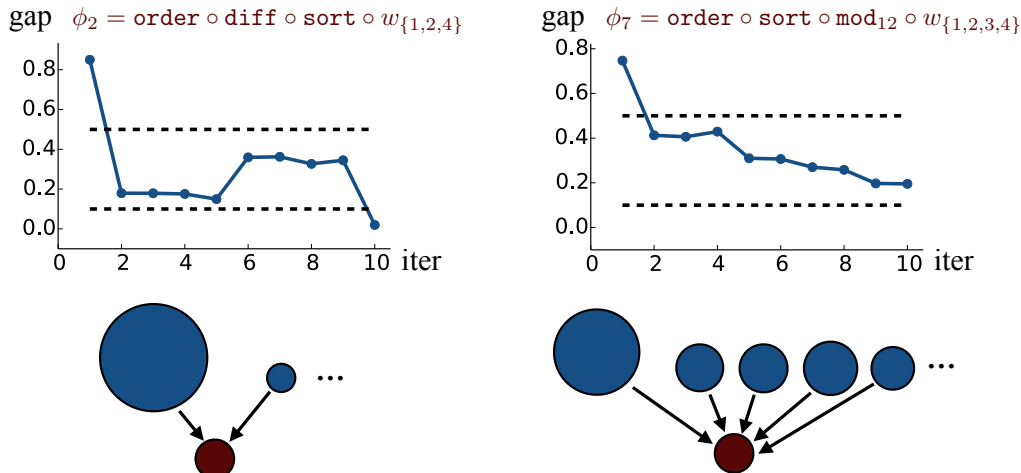


Figure 13.7: Gap trajectories for two features ϕ_2 and ϕ_7 : the dashed black lines show two different satisfactory gaps ($\gamma = 0.5$ and 0.1). The bottom charts show the informationally implied hierarchies: the blue and the red bubbles are implying and implied rules, respectively.

recorded to form a trajectory when cycling the loop. Studying the *vanishing point* of the trajectory reveals the (local) informational hierarchy around the corresponding feature. Taking the second and seventh rule in the slower syllabus for example, we plot their trajectories in Figure 13.7. Both illustrate a decreasing trend¹ for gaps in the corresponding feature spaces. The left figure shows that the second rule is largely but not entirely implied by the first, pointing out the hierarchical structure between the two: the first rule may be considered as the dominant ancestor of the second, which is not conceptually apparent, but informationally implied. Contrarily, the right figure shows that the seventh rule is *not* predominantly implied by the first, which instead is informationally connected to many other rules. However, one could say that it is probably safe to skip both rules in light of a faster pace, since they will eventually be learned fairly effectively (with small gaps) but indirectly.

Part Writing: Adaptive n-grams. Unlike fundamentals which studies sonority independently along the *vertical* direction of the chorale texture, rules on part writing (e.g. melodic motion, chord progression) are *horizontal*, and *context-dependent*. This naturally results in an online learning framework, in which rule extractions are coupled in the writing process, specific to the realization of a composition (context). Context dependence is captured by the multi- n -gram language model, which further leads to the 2D memory pool of features for rule extraction (Section 13.3.4). Consider an example of online learning and

¹Fluctuations on the trajectory are largely incurred by the imperfect solver of the optimization problem.

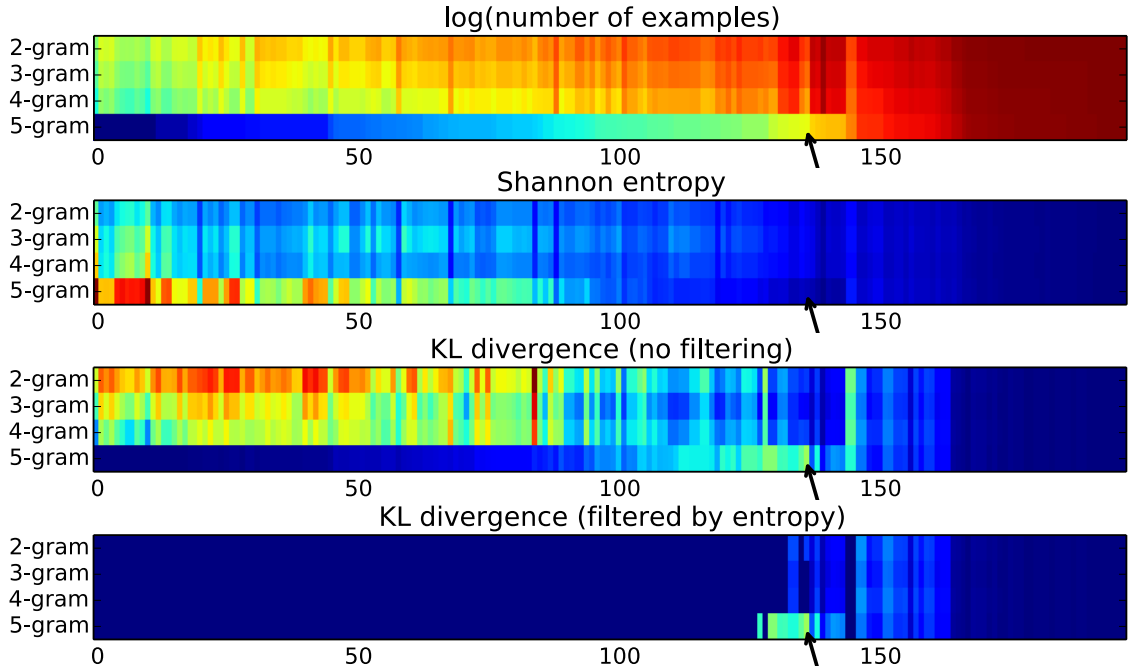


Figure 13.8: The relative performance of the selected rule (pointed) among the pool of all cells in the 2D memory. A desired rule has: higher confidence (measured by the number of examples, brighter regions in the first row), more regularity (measured by Shannon entropy, darker regions in the second row), and larger style gap (measured by KL divergence, brighter regions in the bottom two rows).

adaptive memory selection, where we have the beginning of a chorale:

$$\langle s \rangle, (60, 55, 52, 36), (60, 55, 52, 36), (62, 59, 55, 43), (62, 59, 55, 43), (62, 59, 55, 43), \quad (13.9)$$

and want to learn the probabilistic model for the next sonority. Instead of starting from scratch, MUS-ROVER II launches the self-learning loop with the ruleset initialized by the fundamentals (incremental learning), and considers the 2D memory $N \times \Phi$, for $N = \{2, 3, 4, 5\}$. The first extracted rule is featured by `order` \circ `sort` \circ `mod`₁₂ \circ $w_{\{3,4\}}$. The rule is chosen because its corresponding feature has a large confidence level (validated by the large number of matched examples), a small entropy after being smoothed by Bayesian surprise, and reveals a large gap against the Bach’s style. Figure 13.8 shows the relative performance of this rule (in terms of confidence, regularity, and style gap) to other candidate cells in the 2D memory. Among the top 20 rules for this sonority, 12 are 5-gram, 5 are 4-gram, 3 are 2-gram, showing a long and adaptive dependence to preceding context.

Visualizing Bach’s Mind. With the hierarchical representations in MUS-ROVER II, we are now able to visualize Bach’s music mind step by step via activating nodes in the

DAG of rule features (similar to neuron activations in a brain). The hierarchical structure, as well as the additive activation process, is in stark contrast with the linear sequence of rules extracted from MUS-ROVER I. Figure 13.9 presents a snapshot of the rule-learning status after ten loops, while the student is writing a sonority in the middle of a piece. The visualization makes it clear how earlier independent rules are now self-organized into sub-families, as well as how rules from a new context overwrite those from an old context, emphasizing that music is highly context-dependent.

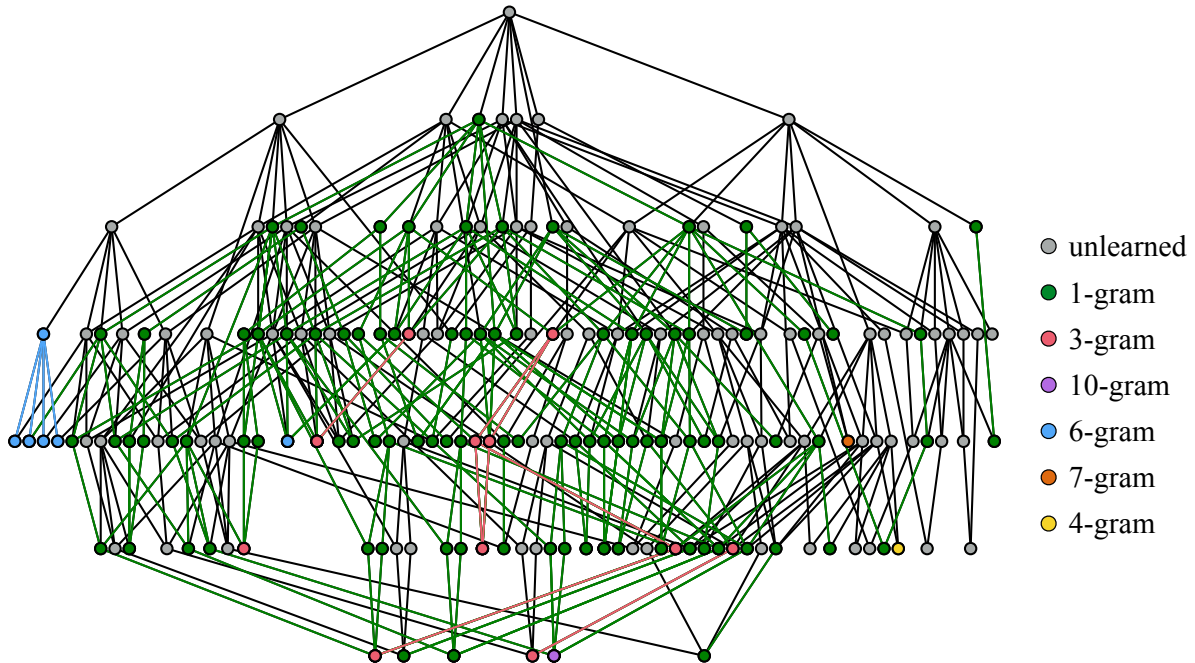


Figure 13.9: Visualization of Bach’s music mind for writing chorales. The underlying DAG represents the conceptual hierarchy (note: edges always point downwards). Colors are used to differentiate rule activations from different n -gram settings. We have enlarged $N = \{1, 2, \dots, 10\}$ to allow even longer-term dependencies.

13.4 MUS-ROVER RB

We implement our unified rule realization and selection framework introduced in Section 10.3 on rule sets from MUS-ROVER II, yielding its variant MUS-ROVER RB for rule breaking. In this section, we demonstrate one experiment, where we exported a set of 16 compositional rules which aims to guide a student in writing the next sonority that follows well with the existing music content.

We show the λ_w -solution path associated with this rule set in Figure 13.10. Again, the

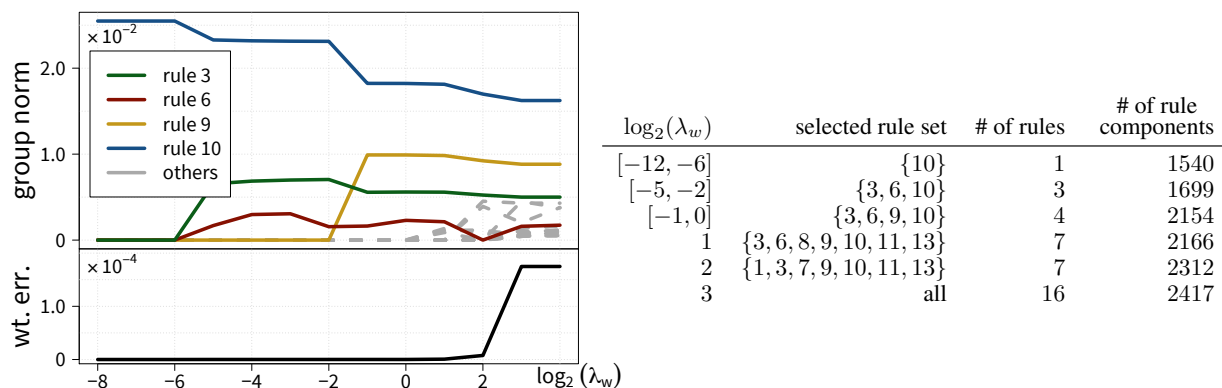


Figure 13.10: The λ_w -solution path obtained from a real compositional rule set.

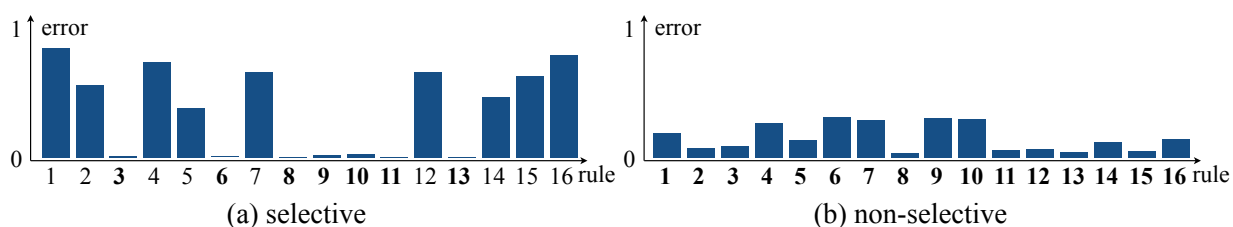


Figure 13.11: Comparison between a selective rule realization ($\log_2(\lambda_w) = 1$) and its non-selective counterpart. The boldfaced x -tick labels designate the indices of the selected rules.

general trend shows the same pattern here: the model turns into a more liberal style (more rules but less accurate) as λ_w increases. Along the solution path, we also observe that the consistent range (i.e. the error-free zone) is wider than that in the artificial cases. This is intuitive, since a real rule set should be largely consistent with minor contradictions, otherwise it will confuse the student and lose its pedagogical purpose. A more interesting phenomenon occurs when the model is about to leave the error-free zone. When $\log_2(\lambda_w)$ goes from 1 to 2, the combined size of the selected rules increases from 2166 to 2312 but the realization error increases only a little. Will sacrificing this tiny error be a smarter decision to make? The difference between the selected rules at these two moments shows that rule 1 and 7 were added into the selection at $\log_2(\lambda_w) = 2$ replacing rule 6 and 8. Rule 1 is about the bass line, while rule 6 is about tenor voice. It is known in music theory that outer voices (soprano and bass) are more characteristic and also more identifiable than inner voices (alto and tenor) which typically stay more or less stationary as background voices. So it is understandable that although larger variety in the bass increases the opportunity for inconsistency (in this case not too much), it is a more important rule to keep. Rule 7 is about the interval between soprano and tenor, while rule 8 describes a small feature between the upper two voices but does not have a meaning yet in music theory. So unlike rule 7 that brings up the important concept of voicing (i.e. classifying a sonority into open/closed/neutral position), rule 8 could

simply be a miscellaneous artifact. To conclude, in this particular example, we would argue that the rule selection happens at $\log_2(\lambda_w) = 2$ is a better decision, in which case the model makes a good compromise on exact consistency.

To compare a selective rule realization with its non-selective counterpart [52], we plot the errors $\|A^{(r)}p - b^{(r)}\|_2$ for each rule $r = 1, \dots, 16$ as histograms in Figure 13.11. The non-selective realization takes all rules into consideration with equal importance, which turns out to be a degenerate case along our model’s solution path for $\log_2(\lambda_w) \rightarrow \infty$. This realization yields a “well-balanced” solution but no rules are satisfied exactly. In contrast, a selective realization (e.g. $\log_2(\lambda_w) = 1$) gives near-zero errors on selected rules, producing more human-like compositional decisions.

13.5 MUS-ROVER RULES VERSUS MUSIC THEORY

We compare probabilistic rules from MUS-ROVER to the basic Western music theory typically taught in a music theory class for music major students. The goal is to use known music theory as a benchmark to see how the produced probabilistic rules correspond to human derived music knowledge: in particular, to see what are covered, what are new, and what are different. However, the goal here is *not* to use known music theory as a ground truth for the purpose of driving MUS-ROVER to reproduce as much as possible. This is because we know in advance three out of the four major differences between laws of music theory and probabilistic rules from MUS-ROVER.

First, in terms of (input) music format, laws of music theory are derived from a complete set of music information whereas probabilistic rules from MUS-ROVER are derived from only MIDI pitches and their durations. The latter is information lossy in at least two places: 1) pitches represented by MIDI numbers are only distinguished up to enharmonic equivalents of those spelled by letter names; 2) many music parameters are currently excluded from MUS-ROVER’s input, e.g. meter, measure, beaming, dynamics.

Second, in terms of (output) rule format, laws of music theory and probabilistic rules from MUS-ROVER are stated in two different ways, with the former being more descriptive and absolute, whereas the latter being more numerical and probabilistic. So, this time regarding rule format, the former is information lossy. For instance, a rule that forbids parallel fifths (which should have been more precisely stated as “consecutive fifths”) may be recovered by a probabilistic rule that assigns a 0.01 probability to the interval class 7 conditioned on a preceding interval class 7. Therefore, while it is possible to “translate” (with information loss) a probabilistic rule (precise) to a rule in known theory (verbal), it is not reasonable to “translate” in the opposite direction and use known rules directly as the true labels like in

a supervised learning scenario (although many earlier rule-based AIs indeed encoded these hard rules to generate somewhat “mechanical” music e.g. Illiac Suite [49]).

Third, in light of purposes, laws of music theory are more intended for pedagogical purposes, thus, are not reflecting the style of a particular data set. For instance, while parallel fifths are banned in homework and exams, they may be widely used in many pop songs; even for our data set of Bach’s chorales (which are supposed to follow the known rules quite well), we see Bach himself did a handful of parallel perfect intervals. On the contrary, probabilistic rules from MUS-ROVER are specific to the input data set. We may certainly find some data sets that follow the known rules quite well (e.g. Bach’s chorales), but find others that break many known rules and even set their own rules.

Keeping these three differences in mind and by further isolating them from the comparison results, we can reveal the remaining differences that are due to the rule-learning process itself. We first compare MUS-ROVER’s feature-induced rules with a complete music theory curriculum taught in the School of Music at the University of Illinois at Urbana-Champaign. To come up with this benchmark, we compiled a comprehensive syllabus of laws of music theory from the course materials used in MUS 502, a graduate theory review class that runs through (in a faster pace) all content covered in a full series of theory classes MUS 101, 102, and 201. This music knowledge is organized as a list of 75 topics indexed by lecture numbers and possibly subtopics within each lecture. On the other hand, probabilistic rules from MUS-ROVER are indexed by abstraction and n -gram as the two coordinates in the 2D memory mentioned earlier in Figure 10.2. The results are summarized in Table 13.6.

<i>Lecture</i>	<i>Music Theory</i>	<i>Abstraction</i>	<i>n-gram</i>	
1	music accents			✗
2	pitch	1-4	1	✓
2	pitch class	16-19	1	✓
2	interval	31-36	1	✓
2	interval class	97-102	1	✓
3	stepwise melodic motion (counterpoint)	1-4	2	✓
3	consonant harmonic intervals (counterpoint)	97-102	1	✓
3	beginning scale degree (counterpoint)	16-19	2	✓
3	ending scale degree (counterpoint)	16-19	2	✓
3	beginning interval class (counterpoint)	97-102	2	✓
3	ending interval class (counterpoint)	97-102	2	✓
3	parallel perfect intervals (counterpoint)	97-102	2	✓

Table 13.6 (cont.)

<i>Lecture</i>	<i>Music Theory</i>	<i>Abstraction</i>	<i>n-gram</i>	
3	directed perfect intervals (counterpoint)			✗
3	law of recovery (counterpoint)	1-4	≥ 3	✓
3	contrapuntal cadence (counterpoint)	1-4, 97-102	2,3	✓
3	melodic minor ascending line (counterpoint)			✗
4	triads and seventh chords	26-30	1	✓
4	triads and seventh chords: quality	140-144	1	✓
4	triads and seventh chords: inversion	113-117	1	✓
5	figured bass	113-117	1,2	✓
5	roman numerals	129-133	1	✓
6	melodic reduction (Schenkerian analysis)			✗
7	passing tone (tones of figuration)	1-4, 134-144	3	✓
7	neighbor tone (tones of figuration)	1-4, 134-144	3	✓
7	changing tone (tones of figuration)	1-4, 134-144	4	✓
7	appoggiatura (tones of figuration)	1-4, 134-144	3	✓
7	escape tone (tones of figuration)	1-4, 134-144	3	✓
7	suspension (tones of figuration)	1-4, 134-144	3	✓
7	anticipation (tones of figuration)	1-4, 134-144	3	✓
7	pedal point (tones of figuration)	1-4	≥ 3	✓
7	(un)accented (tones of figuration)			✗
7	chromaticism (tones of figuration)			✗
8	tonic (function)			✗
8	dominant (function)			✗
8	authentic cadence	1,4,129-133	2,3	✓
8	half cadence	129-133	2,3	✓
9	voice range (four-part texture)	1-4	1	✓
9	voice spacing (four-part texture)	31-41	1	✓
9	voice exchange (four-part texture)	20-25	2	✓
9	voice crossing (four-part texture)	53-63	1	✓
9	voice overlapping (four-part texture)			✗
9	tendency tone (four-part texture)	16-19	1,2	✓
9	doubling (four-part texture)	86-91	1	✓
10	harmonic reduction (second-level analysis)			✗
11	expansion chord			✗
12	predominant (function)			✗

Table 13.6 (cont.)

<i>Lecture</i>	<i>Music Theory</i>	<i>Abstraction</i>	<i>n-gram</i>	
13	phrase model			✗
14	pedal or neighbor (six-four chord)	4,113-117	3	✓
14	passing (six-four chord)	4,113-117	3	✓
14	arpeggiated (six-four chord)			✗
14	cadential (six-four chord)	113-117, 133	3,4	✓
15	embedded phrase model			✗
16	non-dominant seventh chord (function)			✗
17	tonic substitute (submediant chord)			✗
17	deceptive cadence (submediant chord)	129-133	2,3	✓
18	functional substitute (mediant chord)			✗
19	back-relating dominant	129-133	2,3	✓
20	period (I)			✗
21	period (II)			✗
22	period (III)			✗
23	applied chords (I)	129-133	2,3	✓
24	applied chords (II)	129-133	2,3	✓
25	applied chords (III)	129-133	2,3	✓
26	modulation (I)			✗
27	modulation (II)			✗
28	binary form (I)			✗
29	binary form (II)			✗
30	modal mixture			✗
31	Neapolitan	129-133	1	✓
32	Italian sixth chord	140-144	1	✓
32	French sixth chord	144	1	✓
32	German sixth chord			✗
32	Swiss sixth chord			✗
33	ternary form			✗
34	sonata form			✗

Table 13.6: Comparison of probabilistic rules from MUS-ROVER to laws of music theory taught in MUS 502. Details on abstraction IDs are shown in Table 13.7. Red crosses (7) denote topics not recoverable from our adopted music raw representations; blue crosses (18) denote topics not recoverable from our n -gram transitions of abstractions.

Out of the 75 topics in Table 13.6, MUS-ROVER covers 45 of them with a raw coverage

rate of 60%. Note that among the 30 uncovered topics, there are 5 accents-related topics and 2 enharmonically re-spellable chords (German and Swiss sixth chords) which require music inputs beyond MIDI pitches and durations (cf. the first aforementioned difference). This means when considering the learnability of the model itself, we should ignore these 7 topics that are more related to data pre-processing and formatting. This reduces the curriculum to a sublist of 68 topics that are recoverable from our adopted music raw representation, yielding a coverage rate of 66%, i.e. 45 out of 68. Among the 23 missed topics, 18 of them are related to deeper-level temporal abstractions such as harmonic functions (7), key areas (2), and forms (10). These temporal abstractions are, more precisely, *abstractions of transitions*, which are implicitly captured but not explicitly recovered from our current multi-abstraction multi- n -gram language model modeling only *transitions of abstractions*. Since MUS-ROVER currently makes static abstractions only, then excluding these 18 topics requiring dynamic abstractions yields a coverage rate of 90%. The missing 10%, i.e. the 5 missed topics, are tricky ones that require ad-hoc encodings which are not explicitly learnable (but may be implicitly captured to some extent) from our general ACL framework.

Accordingly, the composition of the total $30 = 7 + 18 + 5$ uncovered topics suggest three future directions to raise the learning capacity of the current MUS-ROVER. The immediate next big move is to introduce temporal (or dynamic) abstractions—abstractions of transitions rather than transitions of abstractions—into our ACL framework. Once implemented, this will recover the 18 missing topics in music theory, but more importantly, it will benefit ACL in general. Therefore, we consider temporal abstractions as the next significant improvement for the general ACL framework regarding time-series data in general. Further allowing MUS-ROVER to admit more variable music inputs can potentially address the 7 uncovered topics. Doing this in an ad-hoc way (i.e. in a music-specific way) seems easy, but doing this in a general way is desired (especially considering the goals of ACL) and needs more thoughts. Similarly, seeking a more general framework that can cover the last 5 tricky topics is more challenging and requires more thoughts. Nevertheless, recall that the goal here is not to reproduce what we know: the purpose of ACL is not to replicate human intelligence, but to augment it. So, we may certainly stop after enabling abstractions of transitions, yielding an improved raw coverage rate of 84% (93% for topics from MIDI notes) which is good enough.

<i>Abstraction</i>	<i>Feature</i>	<i>Abstraction</i>	<i>Feature</i>
1	w_1	100	$\text{mod}_{12} \circ \text{diff} \circ w_{2,3}$
2	w_2	101	$\text{mod}_{12} \circ \text{diff} \circ w_{2,4}$
3	w_3	102	$\text{mod}_{12} \circ \text{diff} \circ w_{3,4}$

Table 13.7 (cont.)

<i>Abstraction</i>	<i>Feature</i>	<i>Abstraction</i>	<i>Feature</i>
4	w_4	103	$\text{mod}_{12} \circ \text{diff} \circ w_{1,2,3}$
5	$w_{1,2}$	104	$\text{mod}_{12} \circ \text{diff} \circ w_{1,2,4}$
6	$w_{1,3}$	105	$\text{mod}_{12} \circ \text{diff} \circ w_{1,3,4}$
7	$w_{1,4}$	106	$\text{mod}_{12} \circ \text{diff} \circ w_{2,3,4}$
8	$w_{2,3}$	107	$\text{mod}_{12} \circ \text{diff} \circ w_{1,2,3,4}$
9	$w_{2,4}$	108	$\text{diff} \circ \text{diff} \circ w_{1,2,3}$
10	$w_{3,4}$	109	$\text{diff} \circ \text{diff} \circ w_{1,2,4}$
11	$w_{1,2,3}$	110	$\text{diff} \circ \text{diff} \circ w_{1,3,4}$
12	$w_{1,2,4}$	111	$\text{diff} \circ \text{diff} \circ w_{2,3,4}$
13	$w_{1,3,4}$	112	$\text{diff} \circ \text{diff} \circ w_{1,2,3,4}$
14	$w_{2,3,4}$	113	$\text{sort} \circ \text{diff} \circ w_{1,2,3}$
15	$w_{1,2,3,4}$	114	$\text{sort} \circ \text{diff} \circ w_{1,2,4}$
16	$\text{mod}_{12} \circ w_1$	115	$\text{sort} \circ \text{diff} \circ w_{1,3,4}$
17	$\text{mod}_{12} \circ w_2$	116	$\text{sort} \circ \text{diff} \circ w_{2,3,4}$
18	$\text{mod}_{12} \circ w_3$	117	$\text{sort} \circ \text{diff} \circ w_{1,2,3,4}$
19	$\text{mod}_{12} \circ w_4$	118	$\text{order} \circ \text{diff} \circ w_{1,2,3}$
20	$\text{mod}_{12} \circ w_{1,2}$	119	$\text{order} \circ \text{diff} \circ w_{1,2,4}$
21	$\text{mod}_{12} \circ w_{1,3}$	120	$\text{order} \circ \text{diff} \circ w_{1,3,4}$
22	$\text{mod}_{12} \circ w_{1,4}$	121	$\text{order} \circ \text{diff} \circ w_{2,3,4}$
23	$\text{mod}_{12} \circ w_{2,3}$	122	$\text{order} \circ \text{diff} \circ w_{1,2,3,4}$
24	$\text{mod}_{12} \circ w_{2,4}$	123	$\text{mod}_{12} \circ \text{sort} \circ w_{1,2}$
25	$\text{mod}_{12} \circ w_{3,4}$	124	$\text{mod}_{12} \circ \text{sort} \circ w_{1,3}$
26	$\text{mod}_{12} \circ w_{1,2,3}$	125	$\text{mod}_{12} \circ \text{sort} \circ w_{1,4}$
27	$\text{mod}_{12} \circ w_{1,2,4}$	126	$\text{mod}_{12} \circ \text{sort} \circ w_{2,3}$
28	$\text{mod}_{12} \circ w_{1,3,4}$	127	$\text{mod}_{12} \circ \text{sort} \circ w_{2,4}$
29	$\text{mod}_{12} \circ w_{2,3,4}$	128	$\text{mod}_{12} \circ \text{sort} \circ w_{3,4}$
30	$\text{mod}_{12} \circ w_{1,2,3,4}$	129	$\text{mod}_{12} \circ \text{sort} \circ w_{1,2,3}$
31	$\text{diff} \circ w_{1,2}$	130	$\text{mod}_{12} \circ \text{sort} \circ w_{1,2,4}$
32	$\text{diff} \circ w_{1,3}$	131	$\text{mod}_{12} \circ \text{sort} \circ w_{1,3,4}$
33	$\text{diff} \circ w_{1,4}$	132	$\text{mod}_{12} \circ \text{sort} \circ w_{2,3,4}$
34	$\text{diff} \circ w_{2,3}$	133	$\text{mod}_{12} \circ \text{sort} \circ w_{1,2,3,4}$
35	$\text{diff} \circ w_{2,4}$	134	$\text{diff} \circ \text{sort} \circ w_{1,2}$
36	$\text{diff} \circ w_{3,4}$	135	$\text{diff} \circ \text{sort} \circ w_{1,3}$
37	$\text{diff} \circ w_{1,2,3}$	136	$\text{diff} \circ \text{sort} \circ w_{1,4}$

Table 13.7 (cont.)

<i>Abstraction</i>	<i>Feature</i>	<i>Abstraction</i>	<i>Feature</i>
38	$\text{diff} \circ w_{1,2,4}$	137	$\text{diff} \circ \text{sort} \circ w_{2,3}$
39	$\text{diff} \circ w_{1,3,4}$	138	$\text{diff} \circ \text{sort} \circ w_{2,4}$
40	$\text{diff} \circ w_{2,3,4}$	139	$\text{diff} \circ \text{sort} \circ w_{3,4}$
41	$\text{diff} \circ w_{1,2,3,4}$	140	$\text{diff} \circ \text{sort} \circ w_{1,2,3}$
42	$\text{sort} \circ w_{1,2}$	141	$\text{diff} \circ \text{sort} \circ w_{1,2,4}$
43	$\text{sort} \circ w_{1,3}$	142	$\text{diff} \circ \text{sort} \circ w_{1,3,4}$
44	$\text{sort} \circ w_{1,4}$	143	$\text{diff} \circ \text{sort} \circ w_{2,3,4}$
45	$\text{sort} \circ w_{2,3}$	144	$\text{diff} \circ \text{sort} \circ w_{1,2,3,4}$
46	$\text{sort} \circ w_{2,4}$	145	$\text{order} \circ \text{sort} \circ w_{1,2}$
47	$\text{sort} \circ w_{3,4}$	146	$\text{order} \circ \text{sort} \circ w_{1,3}$
48	$\text{sort} \circ w_{1,2,3}$	147	$\text{order} \circ \text{sort} \circ w_{1,4}$
49	$\text{sort} \circ w_{1,2,4}$	148	$\text{order} \circ \text{sort} \circ w_{2,3}$
50	$\text{sort} \circ w_{1,3,4}$	149	$\text{order} \circ \text{sort} \circ w_{2,4}$
51	$\text{sort} \circ w_{2,3,4}$	150	$\text{order} \circ \text{sort} \circ w_{3,4}$
52	$\text{sort} \circ w_{1,2,3,4}$	151	$\text{order} \circ \text{sort} \circ w_{1,2,3}$
53	$\text{order} \circ w_{1,2}$	152	$\text{order} \circ \text{sort} \circ w_{1,2,4}$
54	$\text{order} \circ w_{1,3}$	153	$\text{order} \circ \text{sort} \circ w_{1,3,4}$
55	$\text{order} \circ w_{1,4}$	154	$\text{order} \circ \text{sort} \circ w_{2,3,4}$
56	$\text{order} \circ w_{2,3}$	155	$\text{order} \circ \text{sort} \circ w_{1,2,3,4}$
57	$\text{order} \circ w_{2,4}$	156	$\text{order} \circ \text{diff} \circ \text{mod}_{12} \circ w_{1,2,3}$
58	$\text{order} \circ w_{3,4}$	157	$\text{order} \circ \text{diff} \circ \text{mod}_{12} \circ w_{1,2,4}$
59	$\text{order} \circ w_{1,2,3}$	158	$\text{order} \circ \text{diff} \circ \text{mod}_{12} \circ w_{1,3,4}$
60	$\text{order} \circ w_{1,2,4}$	159	$\text{order} \circ \text{diff} \circ \text{mod}_{12} \circ w_{2,3,4}$
61	$\text{order} \circ w_{1,3,4}$	160	$\text{order} \circ \text{diff} \circ \text{mod}_{12} \circ w_{1,2,3,4}$
62	$\text{order} \circ w_{2,3,4}$	161	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,2}$
63	$\text{order} \circ w_{1,2,3,4}$	162	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,3}$
64	$\text{diff} \circ \text{mod}_{12} \circ w_{1,2}$	163	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,4}$
65	$\text{diff} \circ \text{mod}_{12} \circ w_{1,3}$	164	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{2,3}$
66	$\text{diff} \circ \text{mod}_{12} \circ w_{1,4}$	165	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{2,4}$
67	$\text{diff} \circ \text{mod}_{12} \circ w_{2,3}$	166	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{3,4}$
68	$\text{diff} \circ \text{mod}_{12} \circ w_{2,4}$	167	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,2,3}$
69	$\text{diff} \circ \text{mod}_{12} \circ w_{3,4}$	168	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,2,4}$
70	$\text{diff} \circ \text{mod}_{12} \circ w_{1,2,3}$	169	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,3,4}$
71	$\text{diff} \circ \text{mod}_{12} \circ w_{1,2,4}$	170	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{2,3,4}$

Table 13.7 (cont.)

<i>Abstraction</i>	<i>Feature</i>	<i>Abstraction</i>	<i>Feature</i>
72	$\text{diff} \circ \text{mod}_{12} \circ w_{1,3,4}$	171	$\text{order} \circ \text{sort} \circ \text{mod}_{12} \circ w_{1,2,3,4}$
73	$\text{diff} \circ \text{mod}_{12} \circ w_{2,3,4}$	172	$\text{order} \circ \text{mod}_{12} \circ \text{diff} \circ w_{1,2,3}$
74	$\text{diff} \circ \text{mod}_{12} \circ w_{1,2,3,4}$	173	$\text{order} \circ \text{mod}_{12} \circ \text{diff} \circ w_{1,2,4}$
75	$\text{sort} \circ \text{mod}_{12} \circ w_{1,2}$	174	$\text{order} \circ \text{mod}_{12} \circ \text{diff} \circ w_{1,3,4}$
76	$\text{sort} \circ \text{mod}_{12} \circ w_{1,3}$	175	$\text{order} \circ \text{mod}_{12} \circ \text{diff} \circ w_{2,3,4}$
77	$\text{sort} \circ \text{mod}_{12} \circ w_{1,4}$	176	$\text{order} \circ \text{mod}_{12} \circ \text{diff} \circ w_{1,2,3,4}$
78	$\text{sort} \circ \text{mod}_{12} \circ w_{2,3}$	177	$\text{order} \circ \text{diff} \circ \text{diff} \circ w_{1,2,3,4}$
79	$\text{sort} \circ \text{mod}_{12} \circ w_{2,4}$	178	$\text{order} \circ \text{sort} \circ \text{diff} \circ w_{1,2,3}$
80	$\text{sort} \circ \text{mod}_{12} \circ w_{3,4}$	179	$\text{order} \circ \text{sort} \circ \text{diff} \circ w_{1,2,4}$
81	$\text{sort} \circ \text{mod}_{12} \circ w_{1,2,3}$	180	$\text{order} \circ \text{sort} \circ \text{diff} \circ w_{1,3,4}$
82	$\text{sort} \circ \text{mod}_{12} \circ w_{1,2,4}$	181	$\text{order} \circ \text{sort} \circ \text{diff} \circ w_{2,3,4}$
83	$\text{sort} \circ \text{mod}_{12} \circ w_{1,3,4}$	182	$\text{order} \circ \text{sort} \circ \text{diff} \circ w_{1,2,3,4}$
84	$\text{sort} \circ \text{mod}_{12} \circ w_{2,3,4}$	183	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,2}$
85	$\text{sort} \circ \text{mod}_{12} \circ w_{1,2,3,4}$	184	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,3}$
86	$\text{order} \circ \text{mod}_{12} \circ w_{1,2}$	185	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,4}$
87	$\text{order} \circ \text{mod}_{12} \circ w_{1,3}$	186	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{2,3}$
88	$\text{order} \circ \text{mod}_{12} \circ w_{1,4}$	187	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{2,4}$
89	$\text{order} \circ \text{mod}_{12} \circ w_{2,3}$	188	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{3,4}$
90	$\text{order} \circ \text{mod}_{12} \circ w_{2,4}$	189	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,2,3}$
91	$\text{order} \circ \text{mod}_{12} \circ w_{3,4}$	190	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,2,4}$
92	$\text{order} \circ \text{mod}_{12} \circ w_{1,2,3}$	191	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,3,4}$
93	$\text{order} \circ \text{mod}_{12} \circ w_{1,2,4}$	192	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{2,3,4}$
94	$\text{order} \circ \text{mod}_{12} \circ w_{1,3,4}$	193	$\text{order} \circ \text{mod}_{12} \circ \text{sort} \circ w_{1,2,3,4}$
95	$\text{order} \circ \text{mod}_{12} \circ w_{2,3,4}$	194	$\text{order} \circ \text{diff} \circ \text{sort} \circ w_{1,2,3}$
96	$\text{order} \circ \text{mod}_{12} \circ w_{1,2,3,4}$	195	$\text{order} \circ \text{diff} \circ \text{sort} \circ w_{1,2,4}$
97	$\text{mod}_{12} \circ \text{diff} \circ w_{1,2}$	196	$\text{order} \circ \text{diff} \circ \text{sort} \circ w_{1,3,4}$
98	$\text{mod}_{12} \circ \text{diff} \circ w_{1,3}$	197	$\text{order} \circ \text{diff} \circ \text{sort} \circ w_{2,3,4}$
99	$\text{mod}_{12} \circ \text{diff} \circ w_{1,4}$	198	$\text{order} \circ \text{diff} \circ \text{sort} \circ w_{1,2,3,4}$

Table 13.7: Abstraction IDs and their inducing features.

From a different source of music theory, we compare MUS-ROVER’s symmetry-induced rules with a complete set of known music operations—the “OPTIC” operations: octave shifts (O), permutations (P), transpositions (T), inversions (I), and cardinality changes (C)—summarized in the book “A Geometry of Music” by the music theorist Dmitri Tymoczko [84].

The results are summarized in Table 13.8, which shows that MUS-ROVER covers the major four types of operations OPTI. More generally, the isometry subgroup $\Sigma''_{music} \leq \text{ISO}(\mathbb{Z}^n)$ introduced in the second example in Section 9.2.3 is indeed a music “closure” of OPTI.

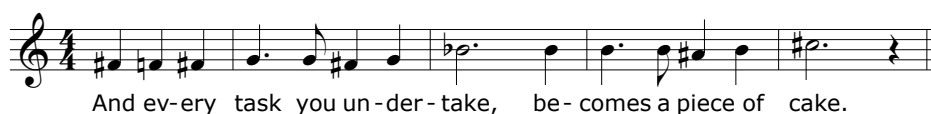
<i>Operation</i>	<i>Music Description</i>	<i>Subgroup</i>
Octave shift	“Move any note into a new octave.”	$\langle \{t_{12e_1}, t_{12e_2}, t_{12e_3}, t_{12e_4}\} \rangle$ ✓
Permutation	“Reorder the object, changing which voice is assigned to which note.”	$\langle \{r_{P(1,2)}, r_{P(2,3)}, r_{P(3,4)}\} \rangle$ ✓
Transposition	“Transpose the object, moving all of its notes in the same direction by the same amount.”	$\langle \{t_1\} \rangle$ ✓
Inversion	“Invert the object by turning it ‘upside down’.”	$\langle \{r_{-I}\} \rangle$ ✓
Cardinality Change	“Add a new voice duplicating one of the notes in the object.”	✗

Table 13.8: Comparison of symmetry-induced abstractions from MUS-ROVER to the OPTIC operations in music. The music descriptions of the operations are quoted from the book “A Geometry of Music” [84].

Chapter 14: MUS-NET: A Crowdsourced Home of Digital Sheet Music

Advances in data-driven study of music are enabling artificial intelligent support for music activities in new ways. These include music concept learning and music education—the main applicational focus of this thesis—as well as new music composition possibilities in general. However, unlike computer vision where there are large-scale repositories of labeled data, there is no centralized music repository that is both easy for a general crowd to contribute to and has sufficiently expressive data for AI algorithms to use.

This chapter introduces MUS-NET, the twin project of MUS-ROVER. It is an online crowdsourcing platform designed for creating and maintaining *digital sheet music*. In particular, this platform enables distributed on-site contributions on making sheet music, and supports direct data access for our automatic music theorists as well as other music AIs in general. The modular design of the platform factors expertise between people and automation, and embeds a self-improving work-correct-validate mechanism. The result is a scalable, reliable music data repository built with amateur, anonymous workers. We demonstrate platform efficacy through results from a music transcription contest with diverse participants; and demonstrate the utility of the repository through AI applications in feature-based music search, optical music recognition (OMR), and automatic music concept learning.



—— A Spoonful of Sugar (from *Mary Poppins*)

Figure 14.1: An excerpt from “A Spoonful of Sugar”.

14.1 MUSIC AI AND MUSIC BIG DATA

Artificial intelligence (AI) has increasingly been applied in music to automate composition, analysis, and other human-centric behaviors traditionally considered creative rather than scientific [109,124,127]. Further, music has progressed beyond its pencil-and-paper roots with new composition tools (e.g. notation software), instrumentation (cf. experimental music), and virtual teachers that are shaping modern music pedagogy [52,128]. To go even further, there is need for a centralized music data repository that is scalable, reliable, and directly accessible by AI algorithms (Figure 14.2: beneficiaries).

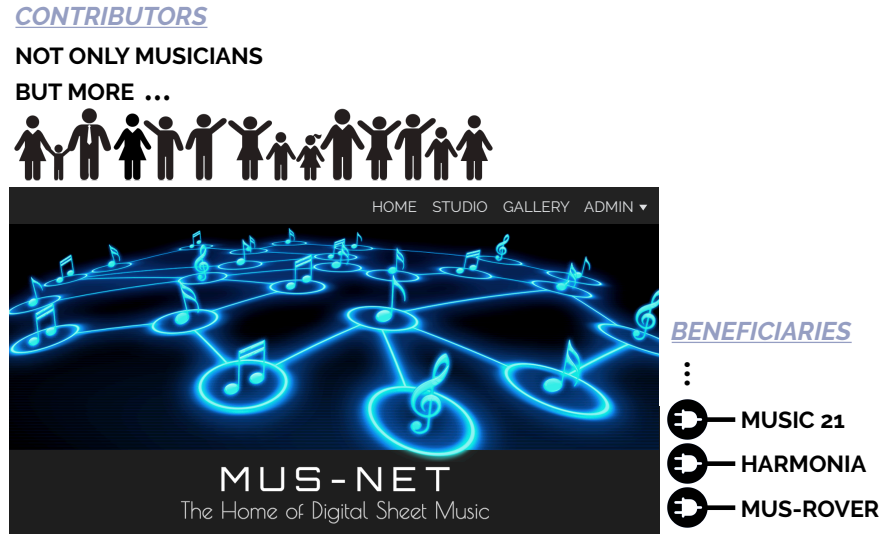


Figure 14.2: MUS-NET, a web-based platform for digital sheet music that admits distributed on-site contributions from the general crowd (contributors), and provides direct plugins to AI applications (beneficiaries).

However, contributing to such a music data repository is too difficult for typical microtask crowd workers, which is essentially a hard human-centric or human-aided OMR [129, 130]. Existing music data repositories typically ask contributors to upload complete music files, whose creation requires both musical knowledge and technological skill. This is in stark contrast to visual data sets like ImageNet [131], created by crowd workers that only needed to parse and label images, since untrained people are able to perform tasks about parts and attributes [132, 133]. We aim to develop a *crowdsourcing* platform for music data transcription to which anyone can contribute (Figure 14.2: contributors), and use this to build an authoritative data repository.

We introduce MUS-NET: a web-based platform that collects, digitizes, and standardizes *sheet music* in the public domain, gathering distributed help from the crowd and directly supporting AI applications. Besides typical music metainformation (e.g. opus number, genre), the resulting repository maintains a digitized symbolic representation—MusicXML—of the core *compositional* ingredients (e.g. pitches and their durations) that capture a composer’s decision at the level of music notes. Importantly, music data in this hierarchical XML format is directly manipulable by machine, eliminating pre-processing steps such as optical music recognition (OMR) for scores or signal processing for sound recordings, to extract the basic frequency and time information of a music note. So, AIs can focus on composition *per se*.

While popular among leading notation software [134], MusicXML is hard to generate, typically involving a steep learning curve for a casual user to become efficient in using any of the

notation software [135]. This results in far less music in this format compared to large music data repositories in general. For manual transcription, the need for reliability and the reality of few professional contributors are in conflict. For automatic transcription, extant commercial converters are of low quality, yet music is sensitive to note-level fluctuations (major and minor chords could be just one semi-tone away) as opposed to largely harmless pixel-level noise in images. As a result, it is not currently realistic to develop a fully automated platform that produces error-free MusicXML. MUS-NET takes a middle path between pure human labor and full automation through careful human-computer interaction (HCI) design.

On the other hand, new opportunities exist in making MusicXML via crowdsourcing. First, MusicXML’s symbolic representation allows factorizing expertise required in making a complete XML file. Crowd workers with no music knowledge can focus on only the visual recognition task regarding localizing notes on music staves (without knowing what these notes are), while more music-related tasks are handled by computational means automatically. This would not be possible if we ask workers to transcribe audio which then would require good aural skills from only qualified workers. Second, compared to visual domain, crowdsourcing in music provides additional modalities to facilitate the transcription as well as the validation and correction of sheet music. For instance, with both the visual (score) and the acoustic (sound) inputs (in contrast to with only one type of input), both laymen and professionals can work more efficiently.

The semi-automatic MUS-NET pipeline comprises modules in a unified architecture (Figure 14.3) that instantiates five principles (not all novel): *AI-friendliness*, *crowd-friendliness*, *scalability*, *reliability*, and *modularity*, and further maximizes automation. The music transcription interface—the central HCI module in the pipeline—is repeatedly used in a so-called transcription-correction-validation loop that incorporates upgradable sub-modules (e.g. auto-checking, auto-completion) to assist both non-experts and professionals in performing high-quality work through distributed collaboration. So, the resulting repository can be self-improved and validated *on site*, while at the same providing direct and reliable access to external AI applications.

In the sequel, Section 14.2 initializes an overview of prevailing music data formats and discusses our choice of data representation. Sections 14.3 and 14.4 present the principles that lead to our design choices for the semi-automatic pipeline, the implementation of the MUS-NET platform, and the resulting repository. We demonstrate in Section 14.5 the efficacy of the platform by presenting results from a music transcription contest held among people with diverse backgrounds; and demonstrate in Section 14.6 the utility of the repository through AI applications in feature-based music search, OMR, and MUS-ROVER. We summarize four main contributions of MUS-NET as follows:

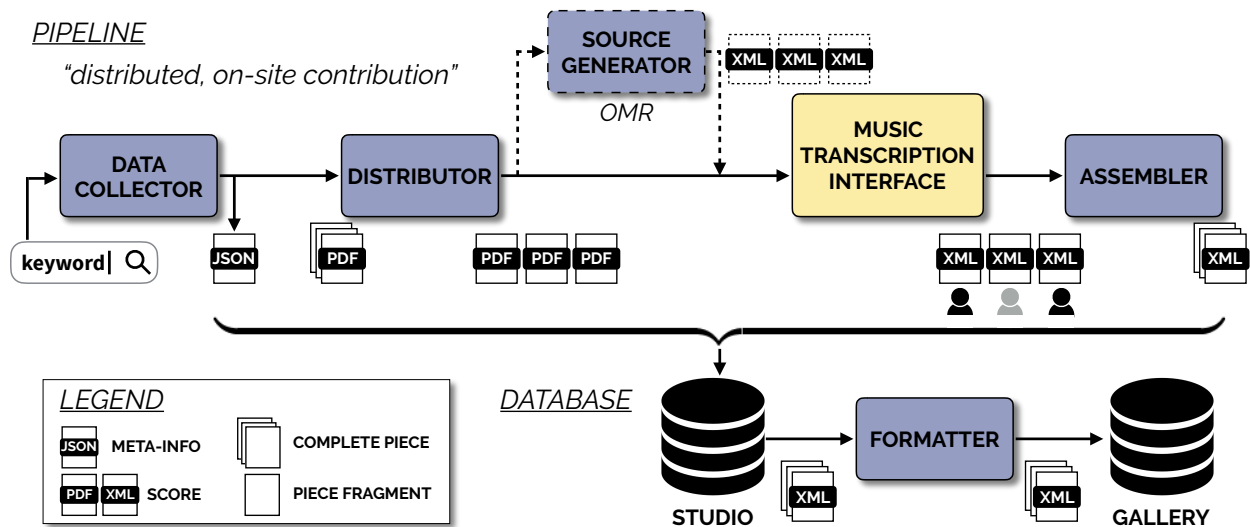


Figure 14.3: The MUS-NET architecture: a modularized pipeline that enables distributed, on-site contribution from the general crowd. It presents the complete workflow of making a reliable MusicXML piece from searching to final formatting. Every piece is a collaborative work among contributors (yellow), amateurs or professionals, and automated units (blue).

1. A growing database to store large music data sets as digital sheet music in the MusicXML format, a hierarchical markup language that is directly machine-readable.
2. Incremental contribution, wherein workers can not only work on a task from scratch but also work on other workers' work and make improvements and validations. This leads to the work-correct-validate cycle embedded in the platform to assure data quality.
3. Domain-specific decomposition of tasks into microtasks to enable distributed contribution and intelligent task-assignment strategies. The idea is to ensure amateurs are not stuck by hard parts and experts are liberated from easy and repetitive ones.
4. Factorization of expertise between people (who focus on fine-grained vision tasks) and automation (which focuses on musical and digital regulations) to lower the bar for the general crowd to contribute.

14.2 OVERVIEW OF MUSIC DATA FORMATS

Music data shares many commonalities with its counterpart in the visual domain: images. Music has note pitches (frequency) and durations (time), whereas images have pixel values (visual frequency) and locations (space); both can have multiple channels for instruments and colors. For digital representation, all of this information is discretized: evenly-spaced

pitches (C, C♯, D, . . .), integer-valued pixels (0,1, . . . , 255), proportional note durations (♩, ♪, ♫, . . .), and 2D image grids. However, unlike an image that is naturally represented by a matrix/tensor, there is no commonly agreed upon music representation: many competitive formats are optimized for different music purposes.

We categorize common music formats into three types based on the level of discretization. Audio waveforms—continuous in both frequency and time—are perhaps the most expressive way to encode acoustic subjects and encompass the largest body of music (or sound in general [136]). MIDI format is another widely used music representation that is discrete in frequency—integer-valued MIDI numbers encoding the pitch (e.g. 69 to A440)—but continuous in time (attack/release time). Lastly, the vast majority of written music is recorded as music scores typically on staff paper: *sheet music*. Also known as the *symbolic representation* of a piece, sheet music is discrete in both frequency and time (e.g. a quarter note C4).

We prefer the symbolic representation in studying composition, not only because historically the large body of written music is engraved in this way, but more importantly it separates *compositional* ingredients from *performance*-related information. While the former literally records a composer’s ideas, the latter incorporates a performer’s personal interpretation of the work. This is a major difference that distinguishes symbolic representation from soundtrack recordings such as audios and MIDI which inseparably fuse composition and performance. Our goal is to build a home for sheet music rather than for recordings (e.g. MusicNet [137]).

14.3 PLATFORM PRINCIPLES

To tackle the challenges involved in supporting AI for music, MUS-NET is built on five underlying principles that manifest in its modular design. These principles are the distinguishing features of the resulting platform and repository, which supports the four main contributions of MUS-NET. In particular, AI-friendliness and scalability, reliability, crowd-friendliness, modularity support the first, second, third, fourth contribution, respectively.

AI-Friendliness (data). To learn composition, we adopt symbolic representation for music. To support an intelligent algorithm to do so, we need a digital representation that is directly accessible by AI [138]. We adopt MusicXML, an XML-based file format that digitally, symbolically represents music notation. Supported by a wide range of music applications (e.g. Finale, MuseScore), this open format has become the standard for exchanging digital sheet music [139]: “Just as MP3 files have become synonymous with sharing recorded music, MusicXML files have become the standard for sharing interactive sheet music.” MusicXML is not only popular among musicians and composers, but also among AI scientists

and engineers that investigate music intelligence [140,141]. This is because MusicXML facilitates direct use in algorithms and programs, saving the effort of low-level data pre-processing and feature extraction (e.g. signal processing for audio) that is irrelevant to composition.

Scalability (repository). In AI, data-driven models typically require large-scale training data sets; the more complex/expressive the model, the more data required [14]. Although there is a huge amount of sheet music, most is archived as either paper manuscripts or their image scans. Among a sample of around 12,000 music scores collected from IMSLP—a virtual library of public domain music scores—only 1/3 are attached with engraving files from notation software, and only part of these are MusicXML or compatible. The conflict between the demand for large AI training data sets and the limited availability of AI-friendly data sets is not solvable by today’s disappointing OMR techniques [142, 143] that poorly convert or even fail to convert scans to MusicXML automatically. This sets scalability as a requirement for MUS-NET as a digital sheet music repository.

Reliability (repository). Music data is very sensitive to note-level fluctuations. A single small alteration of a note can dramatically change the music mode: the contrasting major and minor chord could be only one semi-tone away! Data reliability must be much higher in “symbolic” domains such as music than in “probabilistic” domains like images where pixel-level variation is tolerable [144]. High reliability should not solely rely on worker’s best efforts; MUS-NET embeds a work-validate-correct cycle that can self-validate and self-improve the quality of music data.

Crowd-Friendliness (platform). There are many MusicXML databases: the official website [139] provides a selected listing of sites with sheet music in MusicXML or compatible formats. However, these are largely *static* databases accepting finalized MusicXML, whose targeted contributors are professional digital sheet music makers. The bar to contribute is high and hinders the growth of these static repositories. Contrarily, MUS-NET enables *on-site collaboration* and the attendant forms of intrinsic motivation coming from autonomy, mastery, and connection, in targeting a much wider contributor group (both amateurs and professionals). It is not just a data repository, but rather a *dynamic* web application with a crowdsourcing platform to build the repository. This online platform allows the general crowd to participate in music transcription tasks in a *distributed* way, leveraging the crowd to collaboratively accomplish and validate the work [145,146]. Further, MUS-NET is an integrated platform including searching, processing, conversion, and transcription. So everyone can come to the site and stay on the site to contribute without needing to install external or third-party tools/plugins. In short, MUS-NET pursues a lower cost of contribution with higher efficiency.

Modularity (platform). Besides collaboration among crowd workers, we also consider

collaboration between human labor and automation, manifested in a modular workflow design that seriously considers human-computer interaction. Modularization factors expertise between human and computer, yielding fully-automated expert modules separated from the ones that require simple human effort such as visual recognition [132, 147]. Two major benefits are achieved.

First, domain knowledge is decomposed and squeezed into automated expert systems, removing dependence on music and digital expertise from the general crowd. Rules in music notation and theory are encoded as computer programs into these expert modules or sub-modules—functioning as either independent units or assistive widgets—to assure professionalism from unskilled workers. This divide-and-conquer approach is natural in the modular design, since more and more parts from human labor can be modularized into automated components as new technologies emerge.

Second, modularization supports easy plugs and un-plugs. Every module is a self-contained system that can be easily detached from the MUS-NET pipeline, deployed in other applications, and plugged back in. On the one hand, an automatic module can be upgraded/extended within its own development, yielding the upgrade/extension of the pipeline in an easy, component-wise way. For instance, as OMR advances, one can simply swap in a new OMR tool to yield improved starting points for the transcription task. On the other hand, the non-automatic transcription module can be unplugged and deployed into educational apps or video games. Once plugged back in the pipeline, the transcription tasks are already imperceptibly achieved as byproducts of these third-party music apps. This gives new possibilities to incentivize people to contribute, e.g. through games.

14.4 PLATFORM IMPLEMENTATION

The MUS-NET architecture comprises two databases: *Studio* and *Gallery* for incomplete and well-done pieces, respectively, as well as five self-contained modules integrated into a complete pipeline: the data collector, the distributor and assembler, the source generator, the main music transcription interface, and the formatter (Figure 14.3). Many extant music repositories have the last module that does a final check and formatting of user-provided digital scores (i.e. the tail of the above pipeline), but requires contributors to directly upload well-done MusicXML files (or compatibles) as input. MUS-NET, instead, provides *full support* in the entire process of generating a MusicXML piece, from searching a score to finalizing its format.

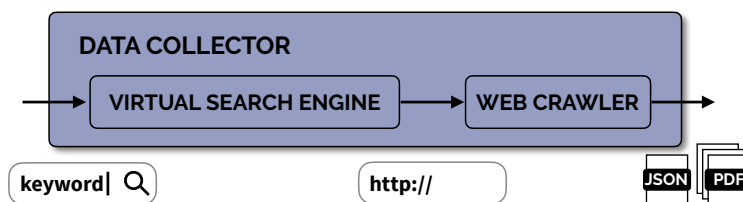


Figure 14.4: Data collector.

14.4.1 Databases: Studio and Gallery

The design of two databases is a distinguishing feature of the MUS-NET pipeline: the added Studio database echoes the platform’s main theme of *on-site contribution*. The term “Studio” indicates the database’s role as a workshop recording all work-in-progress piece fragments and their status. A piece *fragment* refers to one page of a PDF score, which is a unit task assigned to MUS-NET contributors. Different pages of the same piece may be assigned to different contributors, and one contributor can take tasks that do not necessarily combine into a single piece. This *distributed* data structure supports distributed task assignments, which is key to crowd contribution and will be detailed in the distributor module. Contrarily, the term “Gallery” indicates the database’s role as a repository of well-done pieces that may be used by beneficiaries including AI applications.

14.4.2 Data Collector (Automatic)

The MUS-NET pipeline starts with a data collector module that provides on-site search and fetch for a piece’s metainformation and its original (scanned) PDF score. This is a fully automated module that comprises a virtual search engine and a web crawler as two main sub-modules (Figure 14.4).

The search engine takes user-specified keywords of a piece (e.g. work title, composer) as input and returns a link that contains the desired information. Instead of requiring users to temporarily leave the site and search through engines like Google or Bing, the MUS-NET search engine does everything in the backend and keeps the user on site. Automating this process eliminates the technological expertise of using “advanced search” capabilities. For instance, the virtual search engine fires up the search request via Google site:search that narrows down the search domain to a set of pre-specified, authoritative music sites (e.g. IMSLP)—not the entire internet—to assure search quality.

The subsequent web crawler takes a piece’s link as input, and scrapes the webpage for metainformation such as title, composer, and genre, as well as the complete score. The

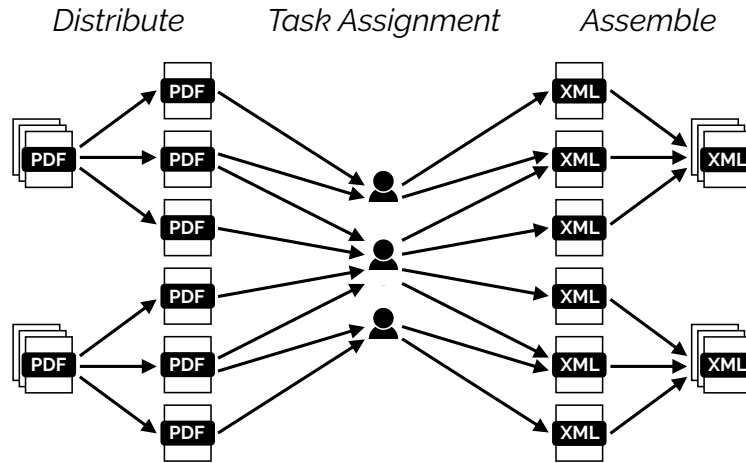


Figure 14.5: Distributed task assignment: a PDF score is decomposed into pages (piece fragments) that are distributed as unit tasks to multiple contributors and assembled into one upon completion. A unit task can be taken by different contributors (cross validation); a contributor can take unit tasks from different pieces (partial contribution).

crawler returns a data package including a JSON and a PDF file for metainformation and score respectively. This initial data package (with JSON and PDF) serves as the *seed* for each piece, from which more advanced operations will be performed in the following parts of the pipeline. Note that a user has the option to directly provide a specific link to the web crawler, if she believes her search expertise surpasses the search engine.

14.4.3 Distributor and Assembler (Automatic)

The distributor and assembler pair, together with the distributed data structure adopted by the Studio database, collectively achieve *distributed* assignment of tasks to the crowd (Figure 14.5). This distributed design allows smart strategies and models concerning a task’s difficulty and/or a contributor’s reliability and can improve the overall efficiency of the entire pipeline. The distributor does two things when communicating with Studio. First, to construct a pool of tasks, the distributor decomposes a complete PDF score page-wise, creates a unit task indexed by the piece and page number, and deposits the unit task into Studio. Second, to make a task assignment, the distributor picks a unit task from Studio per a contributor’s request, and assigns it to the contributor by tagging her user ID to the task in Studio. As the counterpart, the assembler regularly checks the status of every piece in Studio, assembles the piece into a single MusicXML file when all of its XML fragments are ready, and signals that the complete piece is ready for final check.

14.4.4 Source Generator (Automatic)

The source generator is an important add-on module that provides a starting point from which a contributor can simplify the transcription task by correction. The pipeline would still work in the absence of generated sources, requiring a contributor to start all tasks from scratch. This module incorporates OMR tools that convert an input PDF score to MusicXML. There are off-the-shelf software packages such as PDFtoMusic Pro, SharpEye, PhotoScore, SmartScore, capella-scan, and Audiveris, many of which report a high conversion accuracy. However, after trying these products, we discovered that the high accuracies were mostly achieved on PDFs that were exported from notation software (e.g. Finale, MuseScore), which is a little silly: if one has to go through the two-step process of generating a MusicXML file from a PDF exported from a notation software, why not just directly export to MusicXML from the software? Moreover, many of the aforementioned OMR tools are stand-alone software that are difficult to automate and integrate into an existing platform. In light of the performance on scanned PDFs and the ability to integrate, Audiveris is the only OMR tool currently integrated into the source generator. Audiveris is an open source PDF-to-MusicXML converter. For simple scores, it works reasonably well; yet for other cases, it performs poorly or even fails completely.

14.4.5 Music Transcription Interface (Semi-Automatic)

The music transcription interface is the central module in the pipeline that assists a contributor in the main task of transcribing a (scanned) PDF score into its MusicXML format. It is optimized for smooth human-and-computer collaboration requiring less, simpler, but better work from human. The interface allows crowd workers to perform well through a minimal number of clicks/drops and keyboard strokes.

Interface Layout. The main interface is divided into several areas (Figure 14.6). Basic information of the transcription task, e.g. title, composer, and page number of a piece, is displayed at the top of the page; the main tool palette, including music notation cells (right) and control buttons (left), is placed at the bottom. In the middle, two sub-windows are laid side by side: the original PDF to the left, and the editing window to the right. The design of the layout is based on enhancing user experience, respecting principles like Fitts's Law.

Lightweight Design. Excluding the PDF window (middle left), the rest of the interface resembles the typical workbench of a music notation editor. However, there are two major differences. First, MUS-NET's notation editor is an integrated sub-module of the entire pipeline. Under the main theme of *on-site contribution*, this is in stark contrast with many

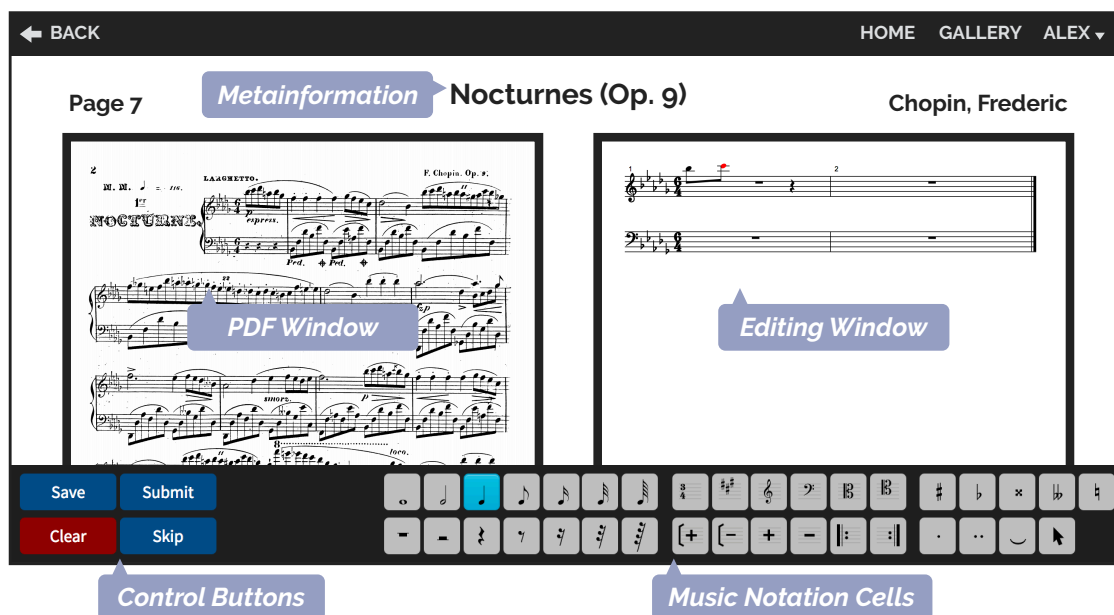


Figure 14.6: Music transcription interface: the main workbench for on-site task completion. The transcription task involves replicating one page of sheet music from its original PDF scan (middle left) into the editing window (middle right) through the provided tool palette (bottom).

other online repositories that requires contributors to either post-install third-party plugins or resort to a notation software offline and come back later to directly submit the generated MusicXML files. Second, MUS-NET’s notation editor is designed to be *lightweight*, thus, a subset of many off-the-shelf notation software in terms of functionality. This is both a virtue and a limitation revealing the tradeoff between expressiveness and ease of use. Considering the steep learning curves and the musician-centric user base of prevailing notation software, we traded expressiveness for ease in our lightweight editor. In addition, targeting the goal of supporting AI to learn the core compositional ingredients, we further simplified the bottom tool palette by excluding notations such as dynamics, tempi, instrumentations.

Transcription-Correction-Validation. The same interface serves three different purposes: transcription, correction, and validation that constitute the *self-improving cycle* for every piece fragment in the Studio database. Accordingly, we categorize MUS-NET contributors into three groups: transcriber, corrector, and validator, and their interfaces only differ slightly (Figure 14.7). The transcriber view and the corrector view are most similar, where variation occurs only in the loading phase of the editing window (middle right). When a transcriber loads the editing window, she will first be prompted with a loading board containing both empty staff paper and many sources from which the transcriber can choose her favorite starting point (Figure 14.7a). When a corrector loads the editing window, the

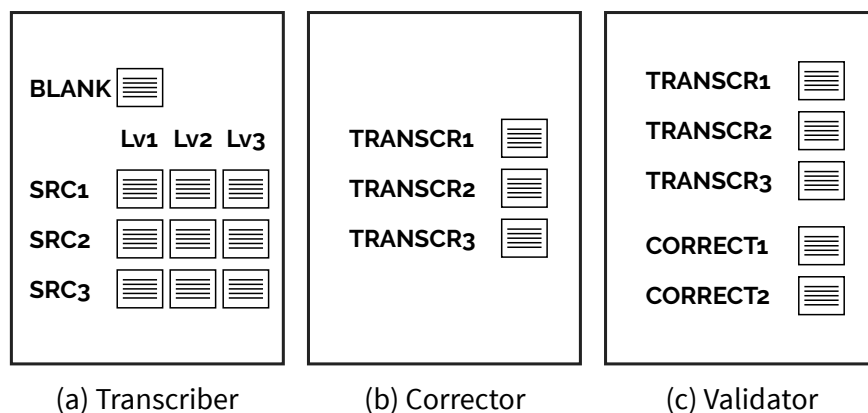


Figure 14.7: Loading boards (of the editing window) in transcription, correction, and validation modes.

content of the loading board is replaced with existing works submitted by the transcribers (Figure 14.7b). The corrector has the option to select the work that appears most reliable, and start improving from there. A validator’s loading board is similar to a corrector’s, but contains works submitted from both transcribers and correctors (Figure 14.7c). In addition, since a validator’s main task is to grade the works but not to edit them, the music notation cells (bottom right) are all disabled, leaving a different set of control buttons (bottom left) for validation. A validator will make a decision in the end to claim if a piece fragment is error-free, thus, ready to finalize.

14.4.6 Formatter (Automatic)

The formatter is the last module in the pipeline, converting input MusicXML to another MusicXML that is both clean and standardized under our proposed criteria. There are many “legal” ways of writing a MusicXML file to record the same compositional ingredients. For instance, some MusicXMLs contain peripheral information on a score’s layout, while some others use sloppy synonyms to encode music elements. To avert this many-to-one relationship between MusicXML files and the actual piece, we propose criteria defining essential elements and standards for formatting MusicXML, which are manifest in the two main sub-modules of the formatter: MusicXML *cleaner* and *standardizer*. After the assembler collects and puts together all the ready fragments into a complete piece in Studio, it sends the piece’s MusicXML file to the cleaner. The cleaner inspects the input MusicXML, and removes all non-essential elements including peripheral information as well as some performance-related information. The standardizer then takes this cleaned version, and returns a unique way of writing the essential elements. After cleaning and standardizing, the final MusicXML is

moved from Studio to Gallery, becoming directly accessible to external AI applications.

Note that the automatic formatter can be a stand-alone module that formats MusicXMLs whenever needed. For instance, it can be used to format OMR outputs in the source generator. In another use case, the pipeline also provides shortcut entry that allows a professional contributor to directly upload a MusicXML file of a complete piece, just like what many other online data repositories do. The MusicXML file uploaded in this way is initially deposited to Studio and automatically marked as ready, then it is directly sent to the formatter for finalization.

14.5 MUSIC TRANSCRIPTION CONTEST

To evaluate the MUS-NET platform, especially the main transcription interface (the only semi-automatic module), we held an online contest with participants recruited from a general crowd of people. The contest asked participants to take transcription tasks from the platform and complete them on site. The participants competed with each other for more and better transcriptions during a given time constraint.

14.5.1 Contest Setup

The online contest lasted for two continuous hours, but could be taken any time at one's convenience. Opening instructions informed the participant of the goal and procedure of the contest, which could be read before and reviewed during the contest. The participant began by clicking the start button located at the end of the instruction page, which took her to the profile page and started the timer.

During the contest, every participant was expected to stay on site and accomplish as many transcription tasks as possible. There are two webpages—profile page and music transcription interface—that a participant could land on and switch between. The profile page (Figure 14.8) is the main portal for taking new tasks. It also displays the identity of the participant and her status in the contest. A participant's status lists a pool of pending tasks, a separate pool of submitted tasks, and another pool of skipped tasks. A pending task is either a new or an incomplete transcription that is saved for later changes; a submitted task is the final version for grading; a skipped task typically indicates challenges due to a participant's knowledge or the platform's capacity, which is (strategically) skipped to save time. While both submitted and skipped tasks do not allow further editing, one can start/continue working on a pending task, which takes the participant to the music transcription interface.

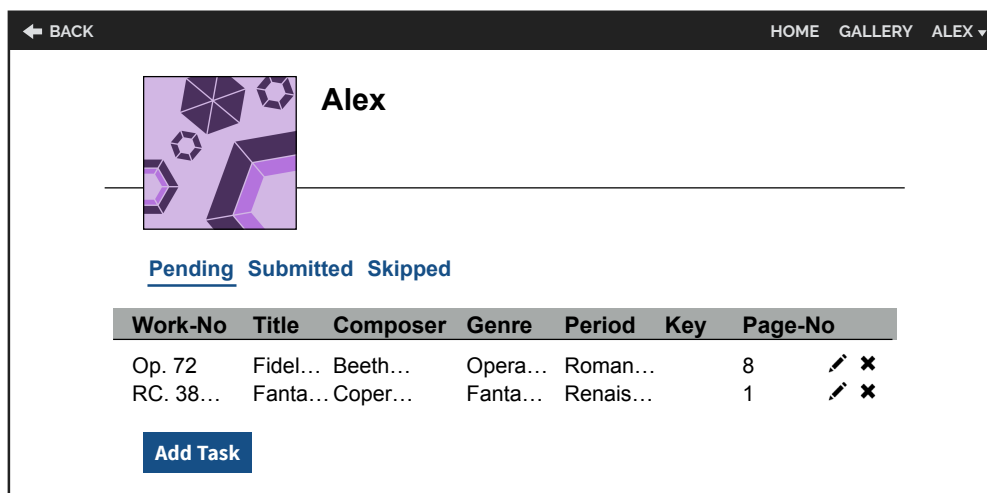


Figure 14.8: Contestant’s profile page: the main portal for getting new tasks and checking status.

The transcription interface (Figure 14.6) is the main site for working on a task. It is equipped with a small tutorial widget located on the navigation bar. The tutorial is a brief introduction of the tools (and their shortcuts if any) on the interface but no music theory or ways/strategies for task completion. It is not required and the participant may or may not spend time on reading the tutorial or part of it; most of the provided functionality can also be explored by simply playing around with the tools. The participant has full freedom in managing her time to save, to submit, and to skip a task, as well as to go back to the profile page for more tasks.

The contest ends when time is up. All participants are invited to complete a post-contest survey to collect demographics as well as comments based on their experience. After the contest, participants can log into MUS-NET again to review what they did in the contest, but no changes can be made.

14.5.2 Evaluation Metric

A participant’s performance in the contest is evaluated by the *speed* and *accuracy* of her transcription. The evaluation metric is made clear to the participant before the contest. We designate a set of basic music elements including notes, accidentals, dots, ties, clefs, key signatures, time signatures, and the like, each of which is worth one point. We also give bonus points for correct music notations that require inference based on music context. For instance, recognizing an incomplete measure (anacrusis) or figuring out the implicit time signature in the middle of a piece is worth more than one point. An expert grader sums

<i>Education</i>	<i>number</i>	<i>Profession</i>	<i>number</i>
In college	15	Engineering	18
Bachelor	15	Arts	11
Master	11	Business	8
PhD	2	Science	5
Others	4	Humanities	4
Prefer not to say	3	Others	4

Table 14.1: Contestants’ education levels and professions.

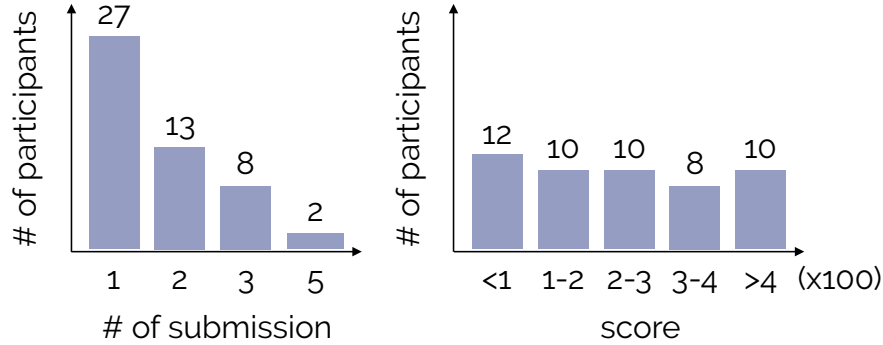


Figure 14.9: Contestants’ overall performances measured by the quantity (left) and quality (right) of their contributions.

points to obtain the participant’s final score, which is a number in the range of $[0, \infty)$.

14.5.3 Results

We report results from 50 contestants (excluding 27 out of 77 total participants due to lack of survey) with diverse education and profession. The balanced distributions indicate that this group of participants is a good representative of the general crowd (Table 14.1).

We analyze participant performance based on both the quantity and quality of their contributions. The frequency distributions of the overall submission rate and performance score are given in Figure 14.9. On average, participants submitted 1.7 tasks and achieved scores of 262.3.

Results are encouraging for a couple of reasons. First, we see the platform is *intuitive* to learn and *easy* to contribute to. Within the two-hour constraint, every participant had to independently explore the platform and figure out her own way/strategy to compete. We purposefully did not provide either training or thorough documentation, having an interface that is intuitive and self-explanatory to participants. The provided tutorial only included a list of tools and shortcuts, whose role was to provide information but not detailed instruc-

tion. Thus, a participant’s performance reflects a combination of her learning ability as well as improved proficiency within the time limit. The fact that the majority of the participants completed a decent amount of work shows that the learning curve for MUS-NET’s contributor is much less steep than that for many extant notation programs (which provide pages of documentation and may require years of training to become proficient).

Second, we see *no zero contribution*. Everyone can make a contribution no matter big or small. We had participants that submitted many incomplete tasks (e.g. one submitted 5 tasks and a high score of 855) and participants that submitted only a few but nearly complete tasks (e.g. another submitted only 2 tasks but also a high score of 600). Since contributors are eventually collaboratively achieving better and better tasks possibly based on other’s work, accepting *partial contribution* allows more contributors to be involved. This is in stark contrast to many extant digital sheet music repositories, where the submission has to be complete, a 0/1 contribution in a non-collaborative process.

Further, given the importance of partial contribution, we demonstrated the efficiency of *incremental contribution*. An incremental contribution refers to a task that is performed from some existing work rather than from scratch. In the contest, existing work refers to sources from the source generator, yet in general, the sources can also be other contributors’ work. Among all submissions with timestamps (timestamps were dropped if completion time is less than 3 seconds), 20% are completed from provided sources with average score 179.3 and average completion time 12.5 min; the remaining 80% are completed from scratch with average score 148.1 and average completion time 42.3 min. This shows that incremental contribution is important in the self-improving cycle of work-correct-validate, since it not only makes contribution easy, but also yields increased accuracy.

Lastly, we report that the average grading/validation time is 8.2 min per task, which is less than transcription and correction time. This emphasizes the point of less work but increased reliability that is manifest in the self-improving mechanism of work-correct-validate.

14.6 AI APPLICATIONS IN ACTION

To evaluate the utility of the MUS-NET repository, we introduce several existing and ongoing projects that build AI applications directly supported by the Gallery database. The applications per se are not the contributions of the thesis, but are used to exemplify use cases of the MUS-NET repository.

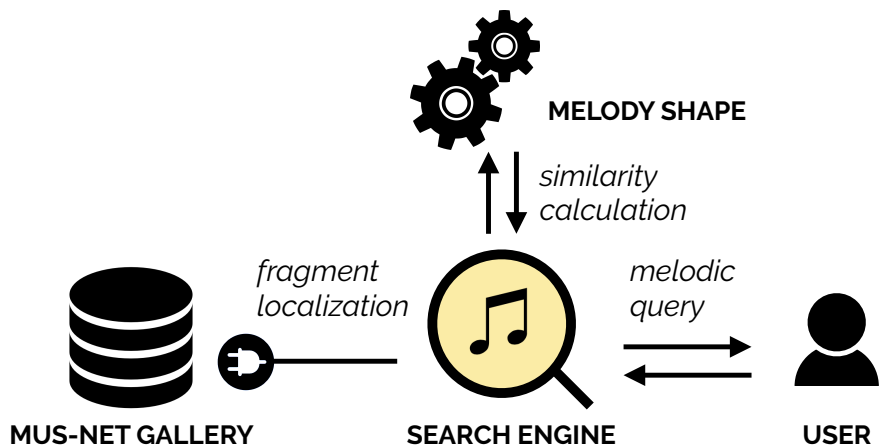


Figure 14.10: Configuration of melody-based search engine.

14.6.1 Compositional-Feature-Based Music Search

Traditional music search engines are text-based, relying on music meta-information as keyword inputs. Now with digital sheet music, we can perform content-based retrieval with multimedia inputs [148]. The Gallery database enables plugging in advanced search engines that rank based directly on features from music compositions. We use MelodyShape [149] as an example for melody-based music search, and turn this algorithm into a search engine (Figure 14.10). The input of the search engine is a query melody in MusicXML format. The user can either upload the XML file directly, or use MUS-NET’s online interface to drop notes in the staff with the corresponding MusicXML automatically generated in the backend. Given the query melody, the search engine looks for melody fragments of equal length (in terms of number of notes) from all parts of all pieces in the Gallery database. It computes similarity scores of the query against all these fragments, and returns a ranking. The ranking not only gives the top- n similar melodies in the database, but also their exact locations.

14.6.2 OMR Training

It is clear from contest results that a good source can greatly increase both the speed and accuracy of the general crowd’s contributions, which further grows the Gallery database. However, this is also reciprocal: a large-scale, reliable MusicXML database can facilitate training better OMR tools by providing ground truth and benchmarks for evaluation. There are known data sets that are prepared specifically for training OMR algorithms [142, 143], but they are largely one-directional rather than achieving a harmonious cycle of mutual benefit as in the MUS-NET platform, where a better data set can help improve OMR, and

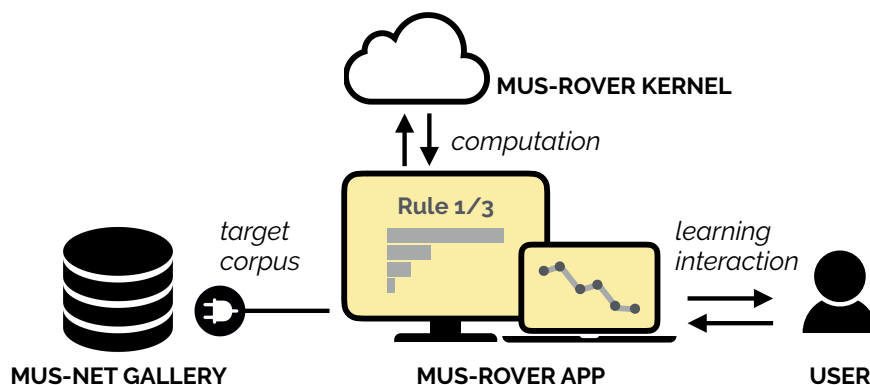


Figure 14.11: Configuration of music education application.

better OMR can help improve the data set.

14.6.3 Automatic Music Rule Learning

As with content-based music search, we can also plug MUS-NET into other AI applications, such as for music education. Recall that in the twin project, MUS-ROVER acts as an automatic music theorist and pedagogue, teaching people step-by-step how to write similar musical works that imitate a target corpus (Chapter 13). There, we built an intelligent music educational bot that offers human-interpretable rules on music composition, which was further wrapped up as a web application that delivers live e-learning experiences in individual/group studies. Unlike many existing e-learning platforms such as Harmonia or Coursera, this app is completely *data-driven* and *personalized*, without requiring hard-coded domain knowledge a priori. The MUS-ROVER web application directly plugs into MUS-NET, communicates with the MUS-ROVER kernel in the backend, and delivers a human-friendly learning interface in the frontend (Figure 14.11). To compile a syllabus for learning, a user selects music pieces from the Gallery database to form the target corpus that she wants to imitate. The app then delivers compositional rules on the fly, revealing the common compositional strategies among the target pieces, customized based on the user’s music background and instant progress.

Chapter 15: Coda

Variants of ACL and ILL formalized in this thesis are also found in some other work on knowledge discovery in topic domains other than music, e.g. science. One example is Schmidt and Lipson’s work on “Distilling Free-Form Natural Law from Experimental Data” [76]. Their learning model can emerge from our self-learning loop as a special case. Instead of probabilistic rules, their rules take the form of mathematical equations that aim to capture important conservations laws. In their specific learning scenario, the input is experimental data from physical systems such as an air-track oscillator and a double pendulum. The output is physical laws (in the form of mathematical equations) of geometric and momentum conservations such as Hamiltonian, Lagrangian, and equation of motion. The teacher (i.e. the discriminative component of the self-learning loop) differentiates partial-derivative-pairs between numerical ones from the data and analytical ones from the student. On the other hand, the student (i.e. the generative component of the self-learning loop) generates candidate functions from which analytical partial derivatives can be derived for the teacher to compare. The two components cooperate with each other to explore the existence of physical laws from the experimental data.

Beyond applications on knowledge discovery, ACL and ILL are also helpful in some other task-specific problems. For instance, localized theories for engineering systems can enhance AI safety in physical engineering systems: if one knows the theory of a system, (s)he can ensure something crazy won’t happen [150]. Classic example is autonomous vehicles, where these complicated systems are hard to theorize about from basic mechanics or dynamics. The same is true for manufacturing systems. Further, ACL is strongly connected to data compression via the same idea of abstracting away invariants, a step that can effectively make big data small. Accordingly, ACL can make supervised learning require much less data by exploiting identified invariances, effectively capturing the small essences of big data.

References

- [1] H. M. Wellman and S. A. Gelman, “Cognitive development: Foundational theories of core domains,” *Annu. Rev. Psychol.*, vol. 43, no. 1, pp. 337–375, 1992.
- [2] T. R. Zentall, E. A. Wasserman, O. F. Lazareva, R. K. Thompson, and M. J. Rattermann, “Concept learning in animals,” *Comp. Cogn. Behav. Rev.*, vol. 3, pp. 13–45, 2008.
- [3] J. M. Mandler, “Perceptual and conceptual processes in infancy,” *J. Cogn. Develop.*, vol. 1, no. 1, pp. 3–36, 2000.
- [4] E. Versace, A. Martinho-Truswell, A. Kacelnik, and G. Vallortigara, “Priors in animal and artificial intelligence: Where does learning begin?” *Trends Cogn. Sci.*, 2018.
- [5] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, “Building machines that learn and think like people,” *Behav. Brain Sci.*, vol. 40, 2017.
- [6] G. F. Marcus, *The Birth of the Mind: How a Tiny Number of Genes Creates the Complexities of Human Thought*. Basic Civitas Books, 2004.
- [7] S. Pinker, *The Language Instinct: How the Mind Creates Language*. Penguin UK, 2003.
- [8] N. Chomsky, *Aspects of the Theory of Syntax*. MIT Press, 2014, vol. 11.
- [9] R. L. Gómez and L. Lakusta, “A first step in form-based category abstraction by 12-month-old infants,” *Developmental Sci.*, vol. 7, no. 5, pp. 567–580, 2004.
- [10] I. Biederman, “Recognition-by-components: a theory of human image understanding,” *Psychol. Rev.*, vol. 94, no. 2, p. 115, 1987.
- [11] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.
- [12] T. Yu, T. Jan, S. Simoff, and J. Debenham, “Incorporating prior domain knowledge into inductive machine learning,” *University of Technology, Sydney*, 2007.
- [13] D. Ferranti, D. Krane, and D. Craft, “The value of prior knowledge in machine learning of complex network systems,” *Bioinformatics*, vol. 33, no. 22, pp. 3610–3618, 2017.
- [14] P. Domingos, “A few useful things to know about machine learning,” *Commun. ACM*, vol. 55, no. 10, pp. 78–87, 2012.
- [15] J.-D. Zucker, “A grounded theory of abstraction in artificial intelligence,” *Phil. Trans. R. Soc. B*, vol. 358, no. 1435, pp. 1293–1309, 2003.

- [16] O. Maimon and L. Rokach, “Introduction to knowledge discovery and data mining,” in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer, 2009, pp. 1–15.
- [17] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.
- [18] U. Grenander and M. I. Miller, *Pattern Theory: From Representation to Inference*. Oxford University Press, 2007.
- [19] U. Grenander and M. I. Miller, “Representations of knowledge in complex systems,” *J. R. Stat. Soc. Ser. B. Methodol.*, vol. 56, no. 4, pp. 549–603, 1994.
- [20] G. Marcus, “Innateness, AlphaZero, and artificial intelligence,” arXiv:1801.05667 [cs.AI], 2018.
- [21] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, “Discovering physical concepts with neural networks,” arXiv:1807.10300 [quant-ph], 2018.
- [22] B. Clark, G. Stein-O’Brien, F. Shiau, G. Cannon, E. Davis, T. Sherman, F. Rajaii, R. James-Esposito, R. Gronostajski, E. Fertig, L. Goff, and S. Blackshaw, “Comprehensive analysis of retinal development at single cell resolution identifies NFI factors as essential for mitotic exit and specification of late-born cells,” *bioRxiv*, p. 378950, 2018.
- [23] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” arXiv:1702.08608 [stat.ML], 2017.
- [24] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. Gershman, and F. Doshi-Velez, “An evaluation of the human-interpretability of explanation,” arXiv:1902.00006 [cs.LG], 2019.
- [25] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, “Explaining explanations: An overview of interpretability of machine learning,” in *2018 IEEE 5th Int. Conf. on Data Sci. and Adv. Anal. (DSAA)*, 2018, pp. 80–89.
- [26] A. Ram and E. K. Jones, *Foundations of Foundations of Artificial Intelligence*. Department of Computer Science, Victoria University of Wellington, 1994.
- [27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.
- [28] M. Hutson, “Basic instincts,” *Science*, vol. 360, no. 6391, pp. 845–847, 2018.
- [29] K. R. Livingston, *Rationality and the Psychology of Abstraction*. Institute for Objectivist Studies, 1998.

- [30] C. Shannon, “The lattice theory of information,” *Trans. IRE Prof. Group Inf. Theory*, vol. 1, no. 1, pp. 105–107, 1953.
- [31] K. Haase, “Discovery systems: From AM to CYRANO,” *MIT AI Lab Working Paper 293*, 1987.
- [32] S. G. Laitz, *The Complete Musician: an Integrated Approach to Tonal Theory, Analysis, and Listening*. Oxford University Press, 2016.
- [33] J. J. Fux, *Gradus Ad Parnasum*. WW Norton & Company, 1965.
- [34] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*. Cambridge University Press, 2002.
- [35] J. B. Fraleigh, *A First Course in Abstract Algebra*. Pearson Education India, 2003.
- [36] D. S. Dummit and R. M. Foote, *Abstract Algebra*. Wiley Hoboken, 2004.
- [37] F. Goodman, *Algebra: Abstract and Concrete*. SemiSimple Press, 2014.
- [38] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [39] H. Li and E. K. Chong, “On a connection between information and group lattices,” *Entropy*, vol. 13, no. 3, pp. 683–708, 2011.
- [40] T. H. L. Chan and R. Yeung, “On a relation between information inequalities and group theory,” *IEEE Trans. Inf. Theory*, vol. 48, no. 7, pp. 1992–1995, 2002.
- [41] R. W. Yeung, *Information Theory and Network Coding*. Springer Science & Business Media, 2008.
- [42] J. Soni and R. Goodman, *A Mind at Play: How Claude Shannon Invented the Information Age*. Simon and Schuster, 2017.
- [43] “Mary Elizabeth Moore ‘Betty’ Shannon,” The Boston Globe, 2017.
- [44] D. A. Grier, *When Computers Were Human*. Princeton University Press, 2013.
- [45] M. L. Shetterly, *Hidden Figures: The American Dream and the Untold Story of the Black Women Mathematicians Who Helped Win the Space Race*. William Morrow and Company, 2016.
- [46] J. R. Pierce and M. E. Shannon, “Composing music by a stochastic process,” *Bell Telephone Laboratories Technical Memorandum MM-49-150-29*, 1949.
- [47] J. R. Pierce, *An Introduction to Information Theory: Symbols, Signals and Noise*. Dover Books, 1980.

- [48] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A proposal for the Dartmouth summer research project on artificial intelligence,” 1955.
- [49] L. Hiller and L. M. Isaacson, *Illiad Suite, for String Quartet*. New Music Edition, 1957.
- [50] H. Yu, L. R. Varshney, G. E. Garnett, and R. Kumar, “MUS-ROVER: A self-learning system for musical compositional rules,” in *Proc. 4th Int. Workshop Music. Metacreation (MUME 2016)*, 2016.
- [51] H. Yu, L. R. Varshney, G. E. Garnett, and R. Kumar, “Learning interpretable musical compositional rules and traces,” in *Proc. 2016 ICML Workshop Hum. Interpret. Mach. Learn. (WHI 2016)*, 2016.
- [52] H. Yu and L. R. Varshney, “Towards deep interpretability (MUS-ROVER II): Learning hierarchical representations of tonal music,” in *Proc. 5th Int. Conf. Learn. Represent. (ICLR 2017)*, 2017.
- [53] H. Yu, T. Li, and L. R. Varshney, “Probabilistic rule realization and selection,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1562–1572.
- [54] J. R. Weinberg, *Abstraction, Relation, and Induction: Three Essays in the History of Thought*. University of Wisconsin Press, 1968.
- [55] F. Giunchiglia and T. Walsh, “A theory of abstraction,” *Artif. Intell.*, vol. 57, no. 2-3, pp. 323–389, 1992.
- [56] L. Saitta and J.-D. Zucker, “Semantic abstraction for concept representation and learning,” in *Proc. Symp. Abstr., Reformul. and Approx.*, 1998, pp. 103–120.
- [57] L. Saitta and J.-D. Zucker, *Abstraction in Artificial Intelligence and Complex Systems*. Springer, 2013.
- [58] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [59] N. Bredeche, Z. Shi, and J.-D. Zucker, “Perceptual learning and abstraction in machine learning: an application to autonomous robotics,” *IEEE Trans. Syst., Man, Cybern. C*, vol. 36, no. 2, pp. 172–181, 2006.
- [60] A. Bundy, F. Giunchiglia, and T. Walsh, *Building Abstractions*. Department of Artificial Intelligence, University of Edinburgh, 1990.
- [61] Y. Bengio, “Learning deep architectures for AI,” *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

- [62] H. Bélai and A. Jaoua, “Abstraction of objects by conceptual clustering,” *Inf. Sci.*, vol. 109, no. 1-4, pp. 79–94, 1998.
- [63] M. Sheikhalishahi, M. Mejri, and N. Tawbi, “On the abstraction of a categorical clustering algorithm,” in *Proc. 12th Int. Conf. Mach. Learn. Data Min. (MLDM 2016)*, 2016, pp. 659–675.
- [64] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley & Sons, 2012.
- [65] R. S. Michalski and R. E. Stepp, “Learning from observation: Conceptual clustering,” *Mach. Learn.*, vol. 1, pp. 331–363, 1983.
- [66] D. H. Fisher, “Knowledge acquisition via incremental conceptual clustering,” *Mach. Learn.*, vol. 2, no. 2, pp. 139–172, 1987.
- [67] R. K. Raman and L. R. Varshney, “Universal clustering,” in *Information-Theoretic Methods in Data Science*, Y. Eldar and M. Rodrigues, Eds. Cambridge University Press, 2019.
- [68] W. M. Rand, “Objective criteria for the evaluation of clustering methods,” *J. Am. Stat. Assoc.*, vol. 66, no. 336, pp. 846–850, 1971.
- [69] U. Von Luxburg, R. C. Williamson, and I. Guyon, “Clustering: Science or art?” in *Proc. 2012 ICML Workshop Unsuperv. Transf. Learn. (UTL 2012)*, 2012, pp. 65–79.
- [70] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.
- [71] L. Rokach and O. Maimon, “Clustering methods,” in *Data Mining and Knowledge Discovery Handbook*, L. Rokach and O. Maimon, Eds. Springer, 2005, pp. 321–352.
- [72] R. M. Cormack, “A review of classification,” *J. R. Stat. Soc. Ser. A. Gen.*, vol. 134, no. 3, pp. 321–367, 1971.
- [73] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 2009, vol. 344.
- [74] S. Y. Oudot, *Persistence Theory: From Quiver Representations to Data Analysis*. American Mathematical Society, 2015, vol. 209.
- [75] C. N. Macrae and G. V. Bodenhausen, “Social cognition: Thinking categorically about others,” *Annu. Rev. Psychol.*, vol. 51, no. 1, pp. 93–120, 2000.
- [76] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *Science*, vol. 324, no. 5923, pp. 81–85, 2009.
- [77] The GAP Group, “GAP – Groups, Algorithms, and Programming, Version 4.9.1,” <https://www.gap-system.org>, 2018.

- [78] L. Hubert and P. Arabie, “Comparing partitions,” *J. Classif.*, vol. 2, no. 1, pp. 193–218, 1985.
- [79] L. Bieberbach, “Über die bewegungsgruppen der euklidischen räume,” *Math. Ann.*, vol. 70, no. 3, pp. 297–336, 1911.
- [80] L. S. Charlap, *Bieberbach Groups and Flat Manifolds*. Springer Science & Business Media, 2012.
- [81] B. Eick and B. Souvignier, “Algorithms for crystallographic groups,” *Int. J. Quantum Chem.*, vol. 106, no. 1, pp. 316–343, 2006.
- [82] H. Zassenhaus, “Über einen algorithmus zur bestimmung der raumgruppen,” *Comm. Math. Helv.*, vol. 21, no. 1, pp. 117–141, 1948.
- [83] V. Felsch and F. Gähler, “CrystCat-a library of crystallographic groups,” *A Refereed Gap*, vol. 4, 2000.
- [84] D. Tymoczko, *A Geometry of Music: Harmony and Counterpoint in the Extended Common Practice*. Oxford University Press, 2010.
- [85] D. Lewin, *Generalized Musical Intervals and Transformations*. Oxford University Press, 2010.
- [86] K. Conrad, “Generating sets,” <http://www.math.uconn.edu/~kconrad/blurbs/grouptheory/genaset.pdf>, 2016.
- [87] R. VanRullen, B. Zoefel, and B. Ilhan, “On the cyclic nature of perception in vision versus audition,” *Phil. Trans. R. Soc. B*, vol. 369, no. 1641, p. 20130214, 2014.
- [88] R. Von Der Heydt, E. Peterhans, and M. R. Dürsteler, “Periodic-pattern-selective cells in monkey visual cortex,” *J. Neurosci.*, vol. 12, no. 4, pp. 1416–1434, 1992.
- [89] A. D. Pape, K. J. Kurtz, and H. Sayama, “Complexity measures and concept learning,” *J. Math. Psychol.*, vol. 64, pp. 66–75, 2015.
- [90] L. R. Varshney, “Mathematical limit theorems for computational creativity,” *IBM Journal of Research and Development*, 2019.
- [91] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 2012.
- [92] L. R. Varshney, “To surprise and inform,” in *Proc. 2013 IEEE Int. Symp. Inf. Theory*, 2013, pp. 3145–3149.
- [93] F. Huszár, “How to train your generative models and why does adversarial training work so well,” <http://www.inference.vc/how-to-train-your-generative-models-why-generative-adversarial-networks-work-so-well-2/>, 2015.

- [94] D. P. Palomar and S. Verdú, “Lautum information,” *IEEE Trans. Inf. Theory*, vol. 54, no. 3, pp. 964–975, 2008.
- [95] R. K. Raman, H. Yu, and L. R. Varshney, “Illum information,” in *Proc. 2017 Inf. Theory Appl. Workshop*, 2017, pp. 1–6.
- [96] J. Wang and J. Ye, “Two-layer feature reduction for sparse-group lasso via decomposition of convex sets,” in *Proc. 28th Annu. Conf. Neural Inf. Process. Syst. (NIPS 2014)*, 2014, pp. 2132–2140.
- [97] T. Hastie, J. Taylor, R. Tibshirani, and G. Walther, “Forward stagewise regression and the monotone lasso,” *Electron. J. Stat.*, vol. 1, pp. 1–29, 2007.
- [98] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, “Least angle regression,” *Ann. Stat.*, vol. 32, no. 2, pp. 407–499, 2004.
- [99] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani, “Strong rules for discarding predictors in lasso-type problems,” *J. R. Stat. Soc. Ser. B. Methodol.*, vol. 74, no. 2, pp. 245–266, 2012.
- [100] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.
- [101] M. Yuan and Y. Lin, “Model selection and estimation in regression with grouped variables,” *J. R. Stat. Soc. Ser. B. Methodol.*, vol. 68, no. 1, pp. 49–67, 2006.
- [102] W. Wang and M. A. Carreira-Perpinán, “Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application,” arXiv:1309.1541 [cs.LG], 2013.
- [103] M. Hong and Z.-Q. Luo, “On the linear convergence of the alternating direction method of multipliers,” *Math. Program.*, pp. 1–35, 2012.
- [104] D. Cope and M. J. Mayer, *Experiments in Musical Intelligence*. AR Editions Madison, 1996, vol. 12.
- [105] J. Biles, “GenJam: A genetic algorithm for generating jazz solos,” in *Proc. Int. Comput. Music Conf. (ICMC)*, 1994, pp. 131–131.
- [106] H. Taube, “Automatic tonal analysis: Toward the implementation of a music theory workbench,” *Comput. Music J.*, vol. 23, no. 4, pp. 18–32, 1999.
- [107] M. Rohrmeier and I. Cross, “Statistical properties of tonal harmony in Bach’s chorales,” in *Proc. 10th Int. Conf. Music Percept. Cogn. (ICMPC)*, 2008, pp. 619–627.
- [108] E. X. Merz, “Implications of ad hoc artificial intelligence in music,” in *Proc. 10th Artif. Intell. and Interact. Digital Entertain. Conf. (AIIDE)*, 2014.

- [109] I. Simon, D. Morris, and S. Basu, “MySong: Automatic accompaniment generation for vocal melodies,” in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI 2008)*, 2008, pp. 725–734.
- [110] M. C. Mozer, “Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing,” *Conn. Sci.*, vol. 6, no. 2-3, pp. 247–280, 1994.
- [111] A. R. Rajanna, K. Aryafar, A. Shokoufandeh, and R. Ptucha, “Deep neural networks: A case study for music genre classification,” in *Proc. IEEE 14th Int. Conf. on Mach. Learn. and Appl. (ICMLA)*, 2015, pp. 655–660.
- [112] D. M. Malioutov and K. R. Varshney, “Exact rule learning via boolean compressed sensing,” in *Proc. 30th Int. Conf. Mach. Learn. (ICML 2013)*, 2013, pp. 765–773.
- [113] S. Dash, D. M. Malioutov, and K. R. Varshney, “Learning interpretable classification rules using sequential rowsampling,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP 2015)*, 2015, pp. 3337–3341.
- [114] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus, “Deep generative image models using a Laplacian pyramid of adversarial networks,” in *Proc. 29th Annu. Conf. Neural Inf. Process. Syst. (NIPS 2015)*, 2015, pp. 1486–1494.
- [115] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, “Adversarial autoencoders,” arXiv:1511.05644 [cs.LG], 2015.
- [116] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proc. 25th Int. Conf. Mach. Learn. (ICML 2008)*, 2008, pp. 1096–1103.
- [117] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [118] G. Desjardins, A. Courville, and Y. Bengio, “Disentangling factors of variation via generative entangling,” arXiv:1210.5474 [stat.ML], 2012.
- [119] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, “Semi-supervised learning with ladder networks,” in *Proc. 29th Annu. Conf. Neural Inf. Process. Syst. (NIPS 2015)*, 2015, pp. 3546–3554.
- [120] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” arXiv:1511.06434 [cs.LG], 2015.
- [121] X. Chen, Y. Duan, R. Houthoofd, J. Schulman, I. Sutskever, and P. Abbeel, “InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets,” arXiv:1606.03657 [cs.LG], 2016.

- [122] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.
- [123] D. Cope, “An expert system for computer-assisted composition,” *Comput. Music J.*, vol. 11, no. 4, pp. 30–46, 1987.
- [124] Google Brain, “Magenta: Make music and art using machine learning,” <https://magenta.tensorflow.org/>, 2016.
- [125] N. Jacoby, N. Tishby, and D. Tymoczko, “An information theoretic approach to chord categorization and functional harmony,” *J. New Music Res.*, vol. 44, no. 3, pp. 219–244, 2015.
- [126] S. Dubnov and G. Assayag, “Universal prediction applied to stylistic music generation,” in *Mathematics and Music*, G. Assayag, H. G. Feichtinger, and J. F. Rodrigues, Eds. Springer Verlag, 2002, pp. 147–159.
- [127] Jukedeck, “Artificially intelligent music composition,” <https://www.jukedeck.com/>, 2012.
- [128] Illiac Software, “Harmonia: Interactive app teaches music theory,” <https://harmonia.illiacsoftware.com/>, 2015.
- [129] L. Chen and C. Raphael, “Human-directed optical music recognition,” *Electron. Imaging*, vol. 2016, no. 17, pp. 1–9, 2016.
- [130] C. Saitis, A. Hankinson, and I. Fujinaga, “Correcting large-scale OMR data with crowdsourcing,” in *Proc. 1st Int. Workshop Digital Libraries for Musicology*, 2014, pp. 1–3.
- [131] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition (CVPR’09)*, 2009, pp. 248–255.
- [132] S. Branson, G. Van Horn, C. Wah, P. Perona, and S. Belongie, “The ignorant led by the blind: A hybrid human–machine vision system for fine-grained categorization,” *Int. J. Comput. Vis.*, vol. 108, no. 1-2, pp. 3–29, 2014.
- [133] L. R. Varshney, P. Jyothi, and M. Hasegawa-Johnson, “Language coverage for mismatched crowdsourcing,” in *Proc. 2016 Inf. Theory Appl. Workshop*, 2016.
- [134] J. Knoder, “The best music notation software,” <http://www.toptenreviews.com/software/home/best-music-notation-software/>, 2017.
- [135] Music Learning Workshop, “Music notation software high end,” <http://www.musiclearningworkshop.com/music-notation-software-high-end/>, 2015.
- [136] H. Dudley, “The Vocoder,” *Bell Lab. Rec.*, vol. 18, no. 4, pp. 122–126, 1939.

- [137] J. Thickstun, Z. Harchaoui, and S. Kakade, “Learning features of music from scratch,” in *Proc. 5th Int. Conf. Learn. Represent. (ICLR 2017)*, 2017.
- [138] K. Chen, A. Kannan, Y. Yano, J. M. Hellerstein, and T. S. Parikh, “Shreddr: Pipelined paper digitization for low-resource organizations,” in *Proc. 2nd Annu. ACM Symp. Comput. for Dev.*, 2012, p. 3.
- [139] MakeMusic, “Musicxml for exchanging digital sheet music,” <http://www.musicxml.com/>, 2011.
- [140] M. Cuthbert, C. Ariza, B. Hogue, and J. W. Oberholtzer, “Music21: A toolkit for computer-aided musicology,” <http://web.mit.edu/music21/>, 2008.
- [141] C.-H. Wei, *Machine Learning Techniques for Adaptive Multimedia Retrieval: Technologies Applications and Perspectives*. IGI Global, 2010.
- [142] P. Bellini, I. Bruno, and P. Nesi, “Assessing optical music recognition tools,” *Comput. Music J.*, vol. 31, no. 1, pp. 68–93, 2007.
- [143] D. Byrd, “Omr systems table,” <http://homes.soic.indiana.edu/donbyrd/OMRSystemsTable.html>, 2007.
- [144] B. Rimoldi, “Beyond the separation principle: A broader approach to source-channel coding,” in *4th Int. ITG Conf. Source and Channel Coding Conf. Rec.*, 2002, pp. 233–238.
- [145] M. Helmstaedter, K. L. Briggman, and W. Denk, “High-accuracy neurite reconstruction for high-throughput neuroanatomy,” *Nat. Neurosci.*, vol. 14, no. 8, pp. 1081–1088, 2011.
- [146] N. Hahn, J. Chang, J. E. Kim, and A. Kittur, “The knowledge accelerator: Big picture thinking in small pieces,” in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI 2016)*, 2016, pp. 2258–2270.
- [147] D. Parikh and C. L. Zitnick, “Human-debugging of machines,” in *Second Workshop Comput. Soc. Sci. Wisdom Crowds*, 2011.
- [148] D. Feng, W.-C. Siu, and H. J. Zhang, *Multimedia Information Retrieval and Management: Technological Fundamentals and Applications*. Springer Science & Business Media, 2013.
- [149] J. Urbano, “MelodyShape: A library and tool for symbolic melodic similarity based on shape similarity,” <https://github.com/julian-urbano/MelodyShape>, 2013.
- [150] N. Kshetry and L. R. Varshney, “Safety in the face of unknown unknowns: Algorithm fusion in data-driven engineering systems,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP 2019)*, 2019.