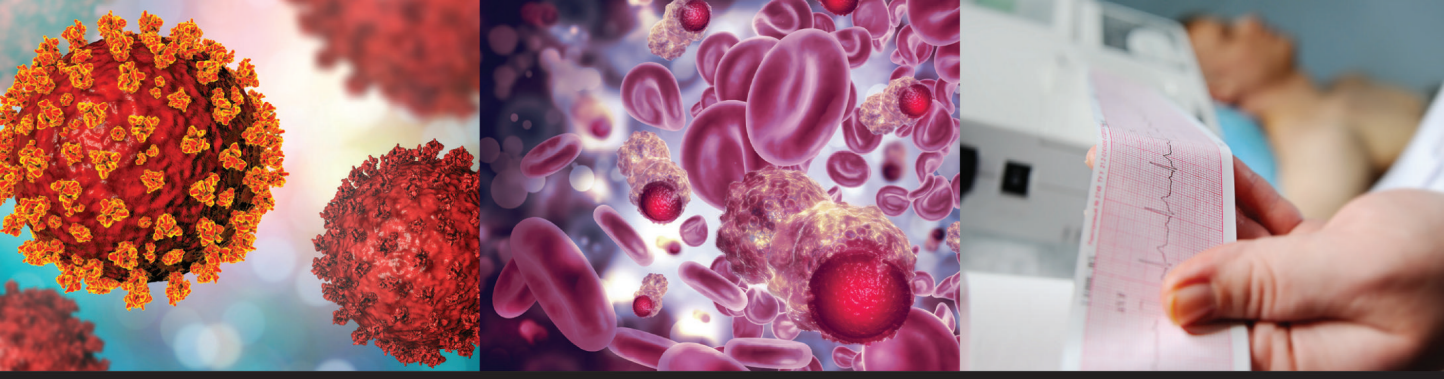
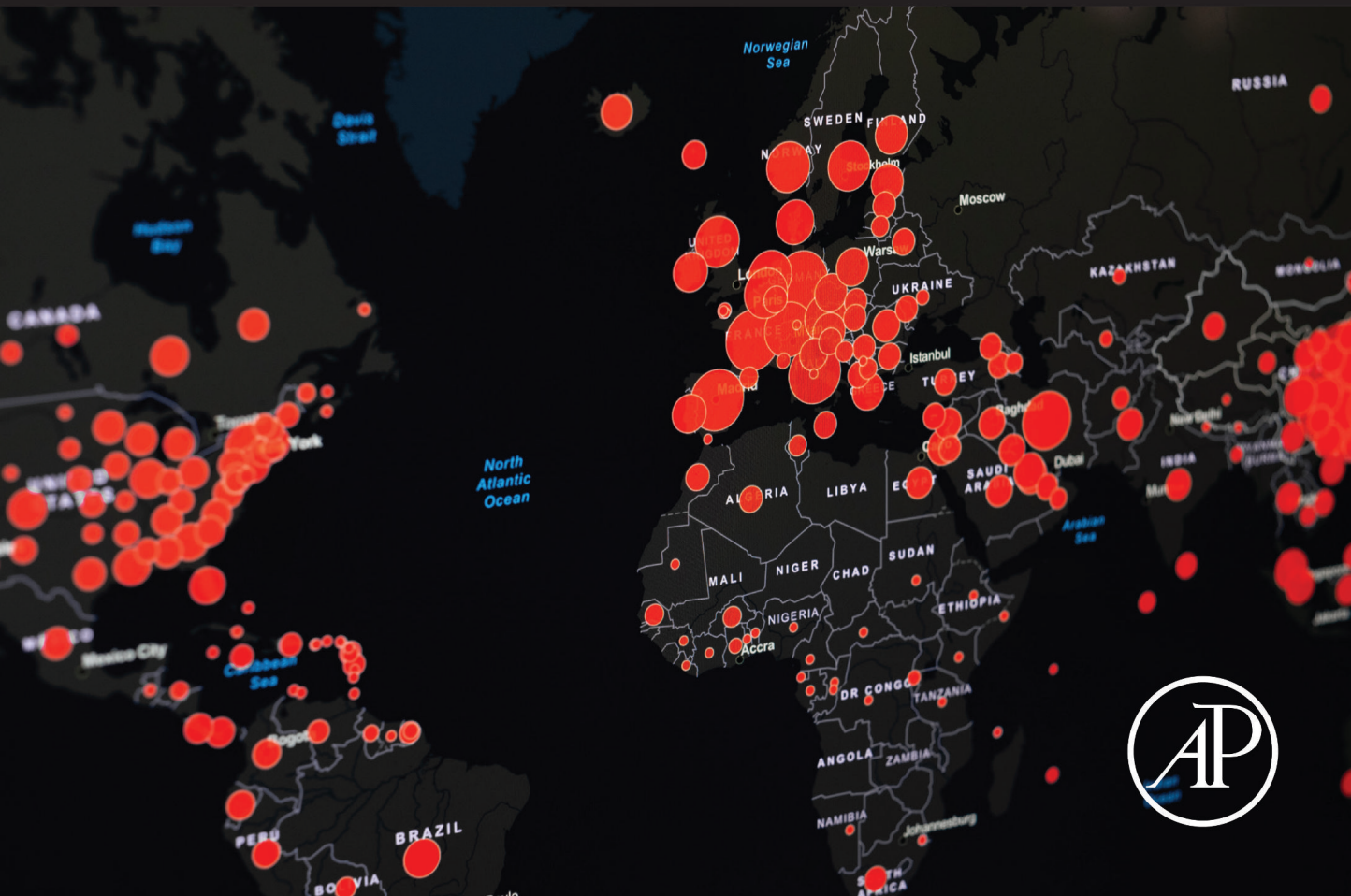


Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care

Amy Neustein and Nathaniel Christen



INNOVATIVE DATA INTEGRATION AND CONCEPTUAL SPACE MODELING FOR COVID, CANCER, AND CARDIAC CARE

This page intentionally left blank

INNOVATIVE DATA INTEGRATION AND CONCEPTUAL SPACE MODELING FOR COVID, CANCER, AND CARDIAC CARE

Amy Neustein
Linguistic Technology Systems
Fort Lee, NJ, United States

Nathaniel Christen
Linguistic Technology Systems
Fort Lee, NJ, United States

ACADEMIC PRESS
An imprint of Elsevier

Academic Press is an imprint of Elsevier
125 London Wall, London EC2Y 5AS, United Kingdom
525 B Street, Suite 1650, San Diego, CA 92101, United States
50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, United Kingdom

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-0-323-85197-8

Publisher: Mara Conner
Acquisitions Editor: Chris Katsaropoulos
Editorial Project Manager: Chiara Giglio
Production Project Manager: Kiruthika Govindaraju
Designer: Christian J. Bilbow

Typeset by VTeX

Contents

III

Conceptual spaces and graph-oriented data-modeling paradigms

7. Multi-aspect modules and image annotation

8. Image annotation as a multi-aspect case study

9. Conceptual spaces and scientific data models

About the authors

Amy Neustein, PhD is CEO and Founder of Linguistic Technology Systems, in Fort Lee, NJ (USA), a think tank for database engineering, scientific computing, and programming language theory. She is the Volume Editor of *Advances in Ubiquitous Computing: Cyber-Physical Systems, Smart Cities, and Ecological Monitoring* (Elsevier 2020) and author/editor of 15 academic books covering a wide range of topics: speech technology, natural language processing, robotics in healthcare, mobile speech, text mining, voice technologies for speech reconstruction and enhancement, signal and acoustic modeling for speech and communication disorders, acoustic analysis of pathologies in infants and children, legal jurisprudence and child health-related issues, forensic speaker recognition, and AI, IoT, Big Data, and Cloud Computing for Industry 4.0. She has authored over 75 articles/chapters/conference papers on this wide panoply of subjects. Dr. Neustein received her PhD in Sociology (with a concentration in sociolinguistics and ethnomethodology) from Boston University. She has served as Editor-in-Chief of the *International Journal of Speech Technology* (Springer) from 2008 till present. She was featured in March 2018 in the SpringerNature "Women in STEM" joint campaign with the United Nations for women in science and technology during Women's History Month. Dr. Neustein serves as Series Editor of *SpringerBriefs in Speech Technology: Studies in Signal Processing, Natural Language Understanding, and Machine Learning* (Springer); Series Editor of two additional book series: *Signals and Communication Technology* (Springer); *Speech Technology and Text Mining in Medicine and Health Care* (de Gruyter).

Nathaniel Christen is Lead Software Architect at Linguistic Technology Systems, in Fort Lee, NJ (USA). He received a BA in Mathematics, with a concentration in theoretical computer science, from Simon's Rock of Bard College in Massachusetts. He later completed his Masters in Cultural Studies at George Mason University in Virginia. He is a doctoral candidate (on leave) at the University of Ottawa in Canada. His doctoral research and dissertation have focused on the interrelation of phenomenology, cognitive linguistics, and the philosophy of science. Mr. Christen has served as a technology advisor for a student/faculty project on the implementation of a document repository serving as a digital archive for the emerging publications of students and faculty members. He served as a Teacher's Assistant at the University of Ottawa, where he taught classes in logic, ethics, and Kantian philosophy. He has programmed extensively in C++, Lisp, and custom languages. He contributed a lengthy chapter on hypergraph-based type theory for software development in a cyber-physical context to Elsevier's *Advances in Ubiquitous Computing*, edited by Amy Neustein.

This page intentionally left blank

CHAPTER

1

Introduction

OUTLINE

1.1 Preface

On May 11, 2021, Dr. Rochelle Walensky, Director of the US Centers for Disease Control and Prevention (CDC), provided testimony to a hearing of the Senate HELP (Health, Education, Labor and Pensions) Committee on the subject of COVID and outdoor activities that proved after the fact to be surprisingly controversial. At issue was how Dr. Walensky defended the CDC's guidance on outdoor activities by indirectly citing a study, which was a statistical outlier, later to be challenged by the same authors whose work she relied upon as a source of up-to-date information. The senators questioning Dr. Walensky focused on SARS-CoV-2 outdoor transmissibility, particularly in the context of when it was "safe" to reopen schools and summer camps. Outside of the committee itself, however, commentators attacked the CDC not only for confusing guidelines, but also for how the CDC director (so it was claimed) misleadingly cited the results of a recent scientific paper—even though (on closer examination) transcripts reveal that her comments were in line with the paper's actual text.

In retrospect, Dr. Walensky's statements did appear to give improper weight to one of multiple studies considered as part of a Systematic Review.[1] Hours after her testimony, one of the Systematic Review's authors used twitter to dispute the CDC director's interpretation of their

[1] *See* https://www.nytimes.com/2021/05/26/briefing/CDC-outdoor-covid-risks-guidelines.html for an overview.

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00005-2

results. The specific issue in contention was an estimation of the percentage of COVID-19 cases that can be traced to outdoor rather than indoor transmission. This actual number is likely to be below 1%.[2] Dr. Walensky, however, cited the Systematic Review paper as claiming a transmission rate of *less than 10%*, a maximum bound derived from the statistical extremum amongst all the articles reviewed therein—instead of a distributive average, a calculation not even attempted in the literature review paper itself.

It is worth noting that in the Systematic Review's very first sentence, the authors report the same 10% upper bound, focusing readers' attention on that specific number, however much the study's text later paints a more granular statistical picture. In an interview with the New York Times, the author who criticized Dr. Walensky's testimony nonetheless argued that she and her co-authors "were very clear we were not making a summary number" with the 10% upper bound, and that their paper was technically a Systematic Review and not (as Dr. Walensky described it) a Meta-Analysis. From a scientific point of reference, as the New York Times correctly stated, "a meta-analysis often includes a precise estimate ... based on the data [whereas] a systematic review is more general."

In short, the author implied that Dr. Walensky was misreading the analytic methods of their paper and as a consequence had presented a quantitative summary that was significantly different from their actual findings. Her 10% estimate was later circulated in the media, creating an impression that outdoor transmissability proportion could be close to such an upper bound, although the relevant statistical distribution in scientific reports actually skews much lower.

Of all the points of contention surrounding COVID-19, this particular controversy stands as little more than a footnote in the annals of US Government response to the pandemic (although confusing statements about outdoor transmissions continue to be cited as a contributing factor in the public's skepticism of the CDC). However minor it may be, though, this episode raises interesting questions about how scientists and policymakers utilize scientific research. After all, we certainly believe that policies (especially in contexts related to medicine and health care) should be informed by empirical data. Yet, how should policymakers interpret research which actually produces empirical data sets? To the extent that many areas of large-scale public concern (certainly COVID-19 is a prime example) engender a number of different research projects, often yielding inconsistent results, how should such diverse data be consolidated into a single model for data-driven policy? Or, is there some threshold of divergence beyond which government officials should simply acknowledge that the science is inconclusive, and defer to scientists to produce more consistent findings before attempting to legitimize policies on empirical terms?

Insofar as multiple research projects often yield conflicting results, citing one specific research work can give a misleading overview of the relevant field as a whole, even if that work was conducted professionally and responsibly. In other words, our criteria for assessing the merits of scientific work inevitably shifts based on the relation of a given research endeavor to others on similar topics. Individual scientists no doubt can do no more than conduct their own research according to disciplined protocols, yielding results which are as conclusive as possible. In this sense well-executed research can be deemed definitive in the specific manner that it was conducted responsibly and in accord with protocols designed to minimize randomness and error. Findings which are *methodologically* sound may still be inaccurate. Whenever multiple research projects address similar themes, to the degree that their findings can be

[2] *See* the specific publication, "Outdoor Transmission of SARS-CoV-2 and Other Respiratory Viruses: A Systematic Review," for details (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7798940).

contrasted, the totality of results among multiple studies should presumably be considered as a context for assessing the claims of any one study individually.

These points may seem obvious, even trivial, but they raise interesting questions under the general rubric of a philosophy of science—research methods are supposed to adhere to rigorous protocols *so that* the corresponding work is empirically persuasive. Empirical science is driven by the belief that we can arrive at factually decisive results by conducting research which is optimized to yield statistically significant results. It is therefore at least philosophically uncomfortable when multiple studies—all done correctly and professionally—yield substantially incommensurate conclusions. Unless specific details in trial design or sample populations (whatever these may be in the research context) are identified—which could explain scientific discrepancies—how can we be confident in the accuracy of individual (even well-implemented) scientific projects when history shows us many cases of equally meritorious research work yielding inconsistent results?

In principle, aggregative compilations of related research work—whether these are called "Systematic Reviews," "Meta-Analyses," or something else—can try to resolve the discrepancies between multiple related publications. Philosophically, however, we should consider whether this simply translates the epistemological uncertainties engendered by contradictory findings onto a different scale. After all, if multiple (well-executed) papers yield divergent results, why should we expect a simply numeric average across all such papers to be more empirically accurate if we have no explanatory mechanism for what caused the discrepancies in the first place?

In the case of Dr. Walensky's testimony, the manner in which she drew conclusions from the Systematic Review inadvertently placed undue weight on one or two specific papers included within that review—according to the New York Times, the 10% upper bound on outdoor versus indoor COVID-19 transmission may have been skewed by one Singaporean study among construction workers, whose working conditions are not representative of outdoor activities by everyday people in general. This demonstrates an indirect version of the fallacy in citing one single paper as a conclusive source when its findings conflict with other similar papers. And yet, data-driven policy has to draw facts from *somewhere* to be empirically grounded in the first place. It is not methodologically evident that some sort of quantitative synthesis of multiple papers should be deemed authoritatively more accurate than the results of one single paper. Is not meta-analysis itself one form of research, which can coexist with other meta-analyses potentially yielding different results in turn, and so on, *ad infinitum*?

Perhaps the best way to get beyond such an impasse is to be more rigorous in how multiple research projects are integrated. To the degree that similar research work yields a spectrum of discrepant results, scientists should actually try to explain those differences, rather than relying on some sort of statistical averaging effect. Thus if findings from one project prove difficult to reproduce, scientists should try to identify the source of this difficulty in either the original work or the attempted replications, or some combination thereof. If one (well-executed) paper appears to be a statistical outlier, scientists should attempt to explain its divergent results.

Of course, some of this interpretive work no doubt occurs, informally and colloquially, as scientists conduct literature reviews or discuss the work of others amongst themselves. However, one could argue that more granular research integration and meta-analysis should be promoted as a formal mechanism within publishing platforms and scientific technology. On this theory, publishers should curate the tools allowing meta-analysis to be performed (more rigorously than today), and the soft-

ware used to generate or analyze scientific data sets should more aggressively implement procedures to integrate data spanning multiple research sources.

In recent years, the scientific community has indeed increasingly embraced open-access paradigms such as FAIR (Findable, Accessible, Interoperable, Reusable) and the Research Object protocol, which we will cite more explicitly in Chapter 2 and elsewhere. These initiatives are intended to make scientific data more transparent and open-access, promoting study replication and/or multi-project syntheses. To be sure, these same paradigms can also apply to meta-analyses themselves. Even when merely aggregating prior research work, there are methodological options which can affect the outcome of a meta-analysis, and the tactics and limitations of aggregative efforts can potentially be modeled systematically. Antecedent research in a meta-analysis (or systematic review) does not necessarily neatly align; some interpretive effort and data-marshaling may be needed to finesse disparate publications and data sets. A study of COVID-19 transmissibility *among construction workers in Singapore*, for example, is not precisely comparable to studies of outdoor SARS-CoV-D infectiousness in general. To have sufficient data in the first place, it may be necessary to rely on research which has certain reciprocal methodological anomalies along these lines.[3] Nevertheless, in each case, researchers can search for a formal mechanism to annotate and systematize the complications which arise from unifying multiple not-fully-compatible research environments into one integrated picture.

[3]As one example, the authors of the controversy-inspiring systematic review described in their paper having to exclude many papers from their analysis to begin with, which perhaps compelled them to retain the anomalous "construction workers" source, with an eye toward keeping a sufficiently large data set.

1.2 Data integration, hypergraphs, and type theory

There is, of course, no magic wand that anyone could wave over the worlds of science and publishing to re-engineer meta-analytic paradigms in a single historical inflection-point. Paradigm-shifts (notwithstanding Kuhnian narratives) tend to happen gradually, and in a decentralized manner. Anyone writing a single chapter or a single book can only hope to predict potential paradigm-shifts or articulate the disciplinary dynamics which could motivate them. Indeed, in this book we are trying to take such an evaluative perspective on the dynamic flow of research applied to the subject of biomedical data acquisition and data-integration practices, in both research and clinical settings.

Plenty of evidence suggests that scientists are starting to take transparent data sharing and research replication seriously. We will consider the idea of "replication crises" in greater detail in Chapter 4; at present, though, we postulate that science has been rocked by a recurring failure to reproduce research results in follow-up studies, even when the original research appears to be well-executed. Separate and apart from the merits of a single research project, a string of similar projects which yield convergent findings have greater scientific weight—so that researchers have an incentive to spur replication studies to solidify the work they themselves publish. It logically follows that an emergent criterion for newly published research is *how well it facilitates* potential reproductions.

This situation does indeed portend a paradigm-shift that has ramifications in numerous scientific and technological domains. The value of having research consolidated through replication can in many contexts outweigh the value of keeping some data or methodological details private—for example, to protect intellectual property—thereby helping to expand the scope of the open-access data publishing ecosystem. Many publishers have likewise embraced

an open-access publishing model, which is also an example of paradigm-shifts that have pushed the scientific community to embrace openness and transparency in sharing research data and methods. Scientists have a reputational stake in exposing their protocols to communal assessment, similar to how computer programmers profit more from open-access code than from closed-source alternatives.[4]

Against this background we can therefore see a certain narrative emerging in how contemporary scientific research is assessed: researchers have an interest in producing work that is (to the degree possible) easily replicable, transparent about methods and/or open vis-à-vis data access, and poised to be integrated with other projects on similar topics. Sometimes these paradigms are explicitly enforced/formally recommended by publishers or funding sources. The Bill and Melinda Gates Foundation, for example, in their "guidelines for authors" (published as one specification within the FAIR-Sharing initiative) essentially requires depositing open-access data sets on popular hosting platforms, such as Dryad, Open Science Foundation, or GitHub, as a precondition for fieldwork grants.[5] Similarly, academic publishers strongly encourage authors to submit data sets to open-access platforms and to reference those platforms via "supplemental materials" and/or "data availability" sections of their articles.[6] In particular, the onus is on authors to justify a *failure* to publicly share research data, such that pairing academic papers with open-access data is intended to be the norm rather than the exception.

Aside from merely observing the industry trends toward freely shared research data and open-access research publications, we can also draw lessons from studying *why* these trends are becoming entrenched. What do scientists actually hope to accomplish by making research data publicly available? One facet of this trend involves double-checking research (and also statistical/computational) methods. The text of an article may condense or summarize findings into a single table or illustration, even just a single number (say, a 1% estimate for the proportion of COVID-19 transmissions which occur outdoors). However, researchers may find that these summaries are more convincing when they can demonstrate their provenance—perhaps by publishing data sets in their entirety rather than just statistical overviews, as well as sharing computer code (through which summarial calculations are attained), data-acquisition protocol descriptions, and so forth. In sum, such transparency in data sharing, among other benefits, can help prevent (or at least retroactively clarify) misinterpretations such as Dr. Walensky's testimonial statements of outdoor COVID-19 infectiousness.

Aside from building trust in their research, data-sharing helps research projects prepare for potential replications, as intimated earlier, as well as for integration with comparable projects. These factors also augment the value of any one individual project.

In short, among the factors driving current data-curation and publishing paradigms is the hope that new research will be attentively reviewed, synthesized with other research, and potentially reproduced/replicated in whole or in part, all of which can make new research more valuable to the relevant scientific community that is in a position to judge the work. In particular, *replication* and *integration* are key goals of new research: the potential for replication (by follow-up studies) and integration (with prior or

[4] Open-source projects get vetted and refined by a larger community of users and fellow developers; commercially licensed versions of open-source projects therefore have a trust factor which closed-source projects frequently lack, making open-source code often more competitive on the open market.

[5] *See* https://gatesopenresearch.org/for-authors/data-guidelines.

[6] *See* https://www.elsevier.com/authors/tools-and-resources/research-data/data-guidelines, for example.

future projects) are among the criteria by which new research is evaluated.

To the degree that these are indeed compelling motivations—researchers strive to model their text, data, and protocols to promote replication and integration—we can anticipate that research methods and tools (including software and computational tools) that amplify these aspects of research's dissemination will be preferred. This can plausibly be deemed a driving factor in how scientific software, data modeling paradigms, and publication technologies will evolve in the immediate future: technologies will come to the fore to the degree that they promote research data integration and replication.

For a concrete example of these issues, consider clinical data-sharing networks such as OMOP (Observational Medical Outcomes Partnership), PCOR (Patient-Centered Outcomes Research Network) or CDISC (the Clinical Data Interchange Standards Consortium). These initiatives promote data integration largely through conventional relational-database mergers, relying on Controlled Vocabularies or table-schema to enforce data field names and table columns. Conversely, Semantic-Web based data-integration projects such as the OBO (Open Biological and Biomedical Ontology) Foundry pursue data integration via more free-form graph structures, regulated by Ontologies or "shape constraints" rather than by static table layouts (we discuss such indirect constraint logics in Chapter 6). A variation on graph-based integration techniques, based on property-graphs rather than semantic web networks, is evident in the University of Pennsylvania "Carnival" project, which we summarize in Chapter 3. Other model-sharing networks emphasize data models prioritizing computational simulations and Object-Oriented representations (we will cite specific examples in Chapter 4). Moreover, individual disciplines within biomedicine and bioinformatics have their own data-sharing protocols, often based on special-purpose file types and domain-specific software, several of which are reviewed in Chapter 2.

In short, researchers working in a biomedical context have multiple options for preparing publicly shared data in preparation for potential integration with other studies (or with replication efforts seeking to reproduce their own specific work). Should data be modeled as relational tables, semantic web style graphs, property graphs, "objects" backed by object-oriented computer code, or as instances of domain-specific data formats? All of these data-representations coexist in contemporary technology with more or less comparable equilibrium, in the sense that no one paradigm dominates the others. How will data-representation strategies evolve in the future?

Insofar as paradigm-shifts in science and publishing are being driven, as we claim, by priorities of data integration and research replication, we can anticipate that technical details such as data meta-models will evolve under pressures from these scientific considerations. Data representations which are conducive to replication and integration will likely become more widely used; less flexible models, such as relational-database technologies, may become deemphasized.

As large-scale initiatives such as OMOP, CDISC, and PCOR evince, the conventional relational database model remains influential. Nonetheless, over the past two generations the Semantic Web—and graph database technology in general—was predicted to substantially displace SQL-style technologies. That has not actually happened (even if Semantic Web formats such as RDF have indeed been widely used). One explanation for why predictions centered on the Semantic Web have fallen flat is that Semantic Web models, intended to be flexible and conceptually realistic, arguably fail by their own standards; this has engendered a trickle of computer scientists criticizing Semantic Web implementations more than motives, and presenting alternative models (such as Conceptual Spaces),

which we will analyze further in Chapters 6 and 9. Meanwhile, graph database technology itself is more general than the Semantic Web alone, and non-RDF formats (more expressive or structurally detailed than labeled graphs as such) have emerged as popular NoSQL database models, including hypergraphs and property-graphs. These technologies, too, may become increasingly influential in structuring how data is modeled for public dissemination and publishing in the future.

In short, we can predict that paradigms such as Conceptual Spaces, hypergraphs, and property-graphs will potentially become more substantial foundations for future data-sharing protocols, which deviate from both Relational-Database and Semantic Web precedents. Work which synthesizes several of these developments, such as hypergraphs and Conceptual Space theory, is therefore especially interesting. One version of a unified hypergraph/conceptual space model has emerged in the context of Quantum Natural Language Processing, and our evaluation of that model's assumptions, potential, and limitations will be an important focus of Chapters 6 and 9.

Supplemental materials for this book will be deposited on the Open Science Foundation, Dataverse, and GitHub.[7] These materials include sample data sets which are encoded and annotated using techniques we discuss in Chapters 6–9. Such data sets are not intended for research purposes themselves, but rather to illustrate ideas about how data sets may be structured, to the degree that scientists really do prioritize expressive data models, microcitations (see below), integration with machine-readable text, and other data-publishing features mentioned in this introduction and elsewhere in the book. The supplemental repository also contains machine-readable text of the book's individual chapters, encoded as separate documents using a special text-representation system designed to facilitate cross-references between publications and data sets. Finally, we include code libraries targeted at the sample data sets, in keeping with design patterns (discussed in later chapters) which assume that data sets will in general be published alongside code libraries that provide functionality to read and manipulate the associated data, including parsers for the data's serialization format.

By sharing code alongside data—optimized as necessary for the specific information comprising a data set—programmers can shift the burden of annotation and documentation to the computer code rather than the data itself. As we will discuss in Chapter 6 and elsewhere, source code presents a richer foundation (as compared to static data models) for pre- and post-condition annotations, requirements engineering, and other technical details that can express scientific theories and research protocols. Equipping data sets with custom-designed code libraries allows the data types uniquely instantiated via those libraries' implementations to serve as models and documentations for the data set's specific profiles; such a type-theoretic foundation, in particular, allows us to examine data set organization in a systematic fashion. We will consider type theory as a data-modeling paradigm in Chapters 5 and 6.

1.3 Philosophy and the semantic web

Data-exchange formats might seem like a mundane scientific topic, part of the minutia of research practice which academics attend to as a matter of professional competence, such as curating biobliographic references, but hardly interesting outside its backstage role. It is curious therefore to consider that the Semantic Web has a colorful philosophical backstory and has had a relatively center-stage position in debates over AI's potentials and limits over whether human intelligence is mechanical enough to be digitally simulated.

[7] For archive locations please visit https://github.com/scignscape/DataIntegration-ConceptualSpaceModeling.

The concept of Semantic Web *Ontologies* has become rather conventionalized such that so-called ontologies tend to serve essentially as Controlled Vocabularies or Taxonomies, constraining the classifications of data within structured information spaces, and the labels used to connect disparate data points. Ontologies are formally rooted in graph database technologies (or information spaces which emulate them, such as the Semantic Web itself). In this context, the principal elements of Ontologies are enumerations of labels that can be attached to graph nodes (providing a classification of the data set embodied by a graph) and graph edges (classifying the kinds of interrelationships that may be asserted between nodes). Ontologies are *Controlled Vocabularies* in the sense that conformant graphs may only utilize node or edge labels proscribed by Ontologies applied to the graph. They are *taxonomies* in that labeled terms may be sorted *hierarchically*: labels can name classification elements which are super- or subkinds, relative to other elements in their respective ontology.

Philosophers understand the term "ontology" to name something of much greater metaphysical weight than just taxonomies on graph data-stores. Notwithstanding that background, the term "ontology" was not chosen by accident; every *domain-specific* Ontology, essentially a data-model applied to empirical data sets, is understood by Semantic Web practitioners to be potentially unified into more general "upper" Ontologies, which have broader (and more philosophical) scope. Upper ontologies aspire to a global classification of "objects" in general, both abstract and concrete, so that in addition to domain-specific concepts and definitions (*carcinoma is a kind of cancer*, say) one has broader metaphysical annotations (cancers are "disease processes," tissues are "spatially extended regions," and so forth). The rationale for this metaphysical superstructure is rooted in AI: ontologies seek to endow scientific and technical data (particularly biomedical data, where ontologies are especially popular) with a conceptual scaffolding analogous to human intelligence, insofar as we instinctively conceptualize objects through the lens of spatiotemporal extension, stasis and change, events and processes, and so forth.

Artificial intelligence is, in fact, a key component of Semantic Web architecture, though in practice the role of AI is less on this metaphysical scale, focusing instead on the use of *axioms* and other Ontology constructions adding structural detail to semantic data models. Ontologies employ axioms and annotations to assert constraints or patterns on how classifications and relations are used. For instance, Ontologies may assert that two relations are inverses (parenthood and childhood, say), or that one relation conceptually implies another—for instance, the relation of two people being *divorced* necessitates that they were previously *married*, such that at some prior point in time the *marriage* relation held. Instead of just sets of graph nodes and edges, then, ontologies add logical structure to graph-form data: the *divorce* relation, for example, demonstrates how relations cluster into logical networks. Any database which represents a divorce-instance, and which is not fundamentally lacking data, would be expected to provide data about the necessarily antecedent *marriage*. We will return to *divorce* as a case study in "multi-part" relationships in Chapter 6.

Ontologies, in short, are not just taxonomies or controlled vocabularies for graph databases; they also introduce axioms and logical constraints on graph-structured data. These extra constructions provide graph-based data models with added expressivity and precision. They also allow graph data structures to be targets for "reasoning engines" and other AI-driven analytics. Reasoning over the Semantic Web is analogous to query-evaluation, but uses methods rooted in Symbolic AI, rather than the query-engine architectures typical of relational (or even NoSQL) databases. This is one sense in which the Semantic Web as a whole represents a

"project" or even ideology closely aligned with AI itself. And the association between AI and the Semantic Web has also been a source of criticism, either from perspectives that are skeptical about the more holistic claims of AI (or "Artificial *General* Intelligence") visionaries, or those which feel that the Semantic Web's fairly modest data-representation paradigms do not fully harness the power of AI (or some combination of the two).

This is some of the milieu in which technical-sounding debates as to optimal data-representation meta-models become invested with unlikely philosophical gravitas. Some of the influential figures in Semantic Web evolution (and criticism) come from the realm of philosophy and humanities/linguistics, not from science (or computer science), such as Barry Smith—a scholar whose earlier work was grounded in phenomenology and Central European philosophy, and who pivoted mid-career to information science, spearheading the OBO Foundry; and Peter Gärdenfors, a linguist who originated Conceptual Space theory and has catalyzed certain counter-narratives critiquing the Semantic Web (mentioned earlier).

At some level, these are relatively minor episodes in intellectual history, and of course the philosophical germination of the Semantic Web has relatively little bearing on a researcher who adopts a specific OBO Ontology, for example, as a structuring device for their data. But philosophical controversies around the Semantic Web help alert us to what is at stake in data-sharing protocols. Why is a vision such as the Semantic Web—a globally synthesized network of knowledge subject to common meta-models that can be concretized in domain-specific standards, potentially synthesizing data from myriad sources—popular? What lies behind its intuitive appeal? The underlying dynamic in debates about (say) Conceptual Spaces versus Ontologies appears to be rooted in scientists' search for data *representations* which seem to intuitively capture the theoretical commitments and conceptual architecture surrounding scientific data, its research origins, and its technical import. In short, scientists seem to consider research data not only as a digital artifact to be shared, but as an embodiment of a given scientific perspective and research environment. Data sets, on this point of view, should *communicate* something about the science and research that produce them, as well as serving the practical goals of moving data from one point to another.

Implicitly, then, in the following chapters when we talk about "data sets" we will usually be considering collections of information which are more than just "raw data." Data sets, specifically, are organized and documented to convey some details about the data set's scientific origins. The data *models* that govern how data sets are encoded thereby need to do more than simply digitize raw data in an unambiguous fashion. Instead, data models have to permit data-set curators to use the organizing principles and annotation mechanisms within each data set as communicative tools representing the appropriate scientific and theoretical background. Such requirements point beyond the formats typically used for data encoding in the past (XML, JSON, HDF, ASDF, and so forth); we will examine criteria and architectures for more expansive and conceptually intuitive formats in later chapters.

1.4 COVID, philosophy, and science

A philosophical discussion which takes the COVID pandemic as a point of departure would be incomplete without acknowledging how some facets of associated public-health policy, such as mask or vaccination requirements, have become politicized. Though COVID-19 serves as a catalyst and magnifier for some of these issues, underlying questions—about the reliability of scientific data, the proper balance between public health and individual rights, and so forth—surely transcend any single disease. It is worth

pointing out several representative controversies where technical issues related to scientific data and publishing are directly relevant.

In the context of vaccinations, for example, consider the noteworthy case of a retracted article, which (not quite appropriately) became widely cited by so-called "anti-vaxxers," that is to say, embroiled in the political machinations of vaccine policy. The contested article (published on June 24, 2021, in the journal *Vaccines*) attempted to quantify the correlation between COVID-19 deaths prevented by vaccination against deaths that *followed* vaccination. The key detail appears to be conflicting interpretations as to whether the authors claim that vaccinations had *resulted in* mortality or merely (acausally) *preceded* fatal outcomes in some circumstances.

After several editors' resignations (some later returned to the journal), *Vaccines* retracted the article, prompting in response a clarification from the authors which appeared to walk back some of their claims. According to *Science Magazine*, the authors conceded that "Currently we only have association ... we never said anything else."[8]

At issue is *first* estimating the number of deaths *prevented by* vaccines (an imperfect process, because one cannot precisely quantify counter-factuals), and *second* distinguishing deaths *caused by* vaccines as opposed to fatal incidents which fit adverse-reaction criteria—so they were *potentially* vaccination side effects—but had other feasible medical explanations as well. Even with that caveat, the authors cited a very low count of fatal incidents (roughly 4 per 100,000) but also (via likely flawed methods) derived a small figure for the hypothetical number of fatal COVID-19 cases prevented by vaccines, on average. Combining both numbers led the authors to summarize that 2 lives are lost for every 3 saved by vaccination, an almost surely inaccurate calculation which was seized upon by anti-vaxxers, who circulated this figure outside its proper context and methodological nascence.

Competent scientists quickly pointed out the article's misleading use of adverse-effect reports as well as their dubious derivation of vaccines' efficacy (based on a single Israeli study comparing vaccinated and unvaccinated cohorts), but the controversy lived on—precisely because scientists questioned how the paper survived peer review in its later-to-be-retracted form to begin with. Even if non-scientists with a political agenda selectively exploited certain details without due consideration for context, peer review did confer an imprimatur of merit on the original work which proved to be premature, but only after having already providing cover for anti-vaccination claims.

A second controversy which presents thought-provoking interpretive questions is the so-called "lab leak" theory, according to which SARS-CoV-2 escaped from a Wuhan research facility, rather than having a zoonotic origin. The preponderance of scientific opinion appears to be that theories of the virus being manually engineered are *plausible*, but also that every epidemiological and genomic factor cited as potential evidence of non-zoonotic origins have alternative (and more mundane) explanations.[9] A notable incident reflecting this dynamic involved Nobel laureate David Baltimore, who was quoted characterizing certain RNA details as "smoking gun" evidence that SARS-CoV-2 was bioengineered. One scientist who led a team investigating possible "genome-tampering," Kristian Andersen (of the Scripps Research Institute), refuted Baltimore's arguments by pointing out that the **CCG** codon accounts for about 3% of the SARS-CoV-2 nucleotides encoding arginine (an amino acid), which is the particular de-

[8] *See* https://www.sciencemag.org/news/2021/07/scientists-quit-journal-board-protesting-grossly-irresponsible-study-claiming-covid-19.

[9] *See*, e.g., https://www.nature.com/articles/d41586-021-01529-3.

tail most often claimed to arise only via human manipulation (SARS-CoV-1 has an even higher rate of this specific encoding, even though little "bioweaponry" speculation accompanied that initial SARS outbreak). Baltimore in an email to *Nature* then moderated his earlier comments, claiming only that "there are other possibilities and they need careful consideration, which is all I meant to be saying."

To some degree, such give-and-take of claims and interpretations is a natural (and healthy) aspect of science, except that (most scientists would probably agree) conducting this exercise through public forums, in a politically charged environment, ends up obscuring the weight of scientific evidence. Insofar as those who wish to advance specific theories or policies (opposing proof-of-vaccine mandates, say, or trying to geopolitically leverage claims that the Chinese government—whether out of malice or incompetence—instigated the pandemic) can find backing for their positions from tendentious peer-reviewed writing and/or decorated experts, the possibility of scientific evidence strongly indicating certain interpretations over others becomes diluted.

At stake here is apparently, again, the contrast between individual studies and the "preponderance of evidence," which in principle should constrain the impact of any single investigation, especially in areas where many research projects overlap and where a particular analysis contradicts the majority of juxtaposable findings. The general maxim that multiple studies yielding similar results carry proportionately greater weight is straightforward, but arguably it remains an open question how to formalize this intuition on a philosophical or meta-scientific level. Notions such as the "weight of evidence" aggregated across multiple studies, or metrics of findings either supporting or deviating from prior research, have not been codified at the meta-analytic scale with the degree of entrenchment that paradigms of "scientific method" are accepted (philosophically as well as operationally) for research projects singularly. One can anticipate that such meta-scientific questions will be explored more rigorously as the overall paradigm of open/transparent data sharing becomes further entrenched.

1.5 How cancer, COVID, and cardiac care may accelerate emerging research trends

It has been suggested that the COVID-19 pandemic may have a lasting impact on many societies partly by accelerating trends which were already latent, such as allowing a greater proportion of office-style work to be performed remotely. In scientific terms, it is hard to overestimate the significance of rapid vaccine development: given that most post-infection interventions have only had limited success in serious cases, it was only via mass vaccinations that fortunate jurisdictions have been able (as of mid-2021) to resume some semblance of normalcy. Moreover, companies were able to develop their vaccines quickly because of considerable research carried out, especially vis-à-vis RNA vaccines, well before 2019. The lesson many scientists and policy-makers may take away from this history is, first, that an infrastructure should be put in place for rapid vaccine testing, development, and production—anticipating future pandemics—and, second, that fundamental science relevant to vaccines can pay dividends in expected ways. In the case of SARS-CoV-2, the science of RNA vaccines become consolidated just before a disease would emerge for which that science would prove invaluable.[10]

One can speculate that the COVID-19 fallout will also leave a residual effect in areas such as scientific publishing, research data management, and the interface between science and

[10] *See* https://www.statnews.com/2020/11/10/the-story-of-mrna-how-a-once-dismissed-idea-became-a-leading-technology-in-the-covid-vaccine-race.

public policy, for reasons already intimated. We have cited examples of civil administrators inciting unexpected controversy even with recommendations that are rooted in up-to-date science; of peer-reviewed (and at least superficially credible) research getting adopted by anti-vaxxers with an agenda that is *prima facie* fundamentally anti-scientific; and of a Nobel laureate essentially dialing back relatively informal (but vaguely sensationalist) comments that fed into the spy-novelesque intrigue of germ warfare.

Trying to reach conclusions from such disparate events may connote a fallacy of some homogeneous "publishing industry" or "scientific community," or "academic establishment," all of which are of course part of a diverse and decentralized social/commercial milieu; nevertheless, some cautious observations may be warranted.

One plausible point of argument is that the proliferation of scientific work *along with* the emergence of data-sharing and data-curation paradigms *jointly* imply that the meaning and notion of "peer review" is noticeably evolving. In this context it is worth referencing analyses such as those by Todd Carpenter (Executive Director of the National Information Standards Organization) on the very nature of peer review applied to *data* rather than *documents*.[11] As this (and similar) studies point out, the process for assessing published data sets—if we stay in the context of peer-review for conventional scientific *literature*—is much less standardized than for publications themselves.

In particular, the fact that an author transparently presents research data as supplemental materials to scientific writings may check one box in a text's favor; however, reviewers might not then proceed (when vetting submissions in general) to closely examine the data sets themselves, with an eye toward measuring them against disciplinary norms. Some scientific groups have accordingly proposed "quality standards" based on factors such as ease of data/code reuse, thoroughness of meta-data, adherence to data-sharing protocols, and the relevance of data to its associated articles. Such standards may then influence how scientific work is received by government officials or the public at large, because the scientific community could indicate whether publications have been developed in the context of high-quality (or, conversely, sub-par) data-curation, adjoining that assessment to other factors (such as peer review or journal reputation) that individuals or policy-makers use to construe scientists' merit in the eyes of other scientists.

Moreover, practically and logistically speaking, we should recognize that data-curation is time-consuming and, in its most rigorous forms, embodies disciplinary knowledge that stands apart from natural sciences themselves. To the degree that diligent data-curation expectations serve as one criterion of research merit, we can envision an evolution in how research teams and/or projects are set up, with proper time and resources allotted to data-management concerns alongside other priorities (those implicated in data-acquisition to begin with).

One could, specifically, envision a gradual conceptual shift away from reading research papers as "static" documents, insofar as data sets (and, where applicable, research code) may be able to cycle through multiple versions, even after publications appear in final form. Version-control systems and branch/clone technologies can allow scientists to keep track of how data and/or code associated with a specific project is evolving while still maintaining the integrity of data sets as citable assets. In this context, scientists may increasingly perceive conventional publications as summaries rather than the substantial core of their research work, and place equal weight on Research Object style resources that can evolve more flexibly/dynamically and over which they have greater editorial control. Likewise, interactive data sets—which can embed multi-media visualization capabilities be-

[11] *See* https://scholarlykitchen.sspnet.org/2017/04/11/what-constitutes-peer-review-research-data.

yond the scope of ordinary print documents (e.g., hooking up a publisher's portal with radiology image data so that readers, while reading an article or book, could see in an engaging 3D format a radiographic image of cardiac tissue, an image of differentiated cancer cells, or an image of ground-glass opacity of lung tissue in COVID patients)—might become esteemed as pedagogic tools helping researchers to exposit their theories and methods (we revisit this point in greater detail in Chapter 4).

Although such a shift in priorities could potentially complicate the concept and process of peer review—because the "work" to be assessed becomes more of a moving target—acknowledging the evolutionary nature of research projects may help restore public confidence in assessments of scientific merit, because it would signal the relevant industries' acknowledgment that appraisals of scientific work are inherently provisional and ongoing. Cases such as *Vaccines*'s retracted article actually reveal science appropriately correcting itself; however, such lingering controversies suggest that non-scientists entertain a misguided conception that research can be crisply sorted into bins of "merit" and "demerit," a simplification that has no basis in science itself.

One may also anticipate the emergence of more rigorous meta-models for scientific data—employing constructions such as type theory or Object Models to classify the basic units of data sets, insofar as these comprise resources with their own histories and quality-control standards. For example, "microcitations" are understood to link natural-language texts with *parts* of data sets, but this definition is inexact: what is a "part" in this context (one record, data-point, measurement, calculation, etc.)? To standardize data-curation assessments, scientists need to define basic formulations such as data sets' version-history (e.g., what are the units of change as a data set gets updated) and overlap/reuse (e.g., what are the units of content that could adjudicate priority claims, plagiarism, impact factor, and so forth)? We explore type-theory in the data curation context (mostly in Chapters 5 and 6) partly because these structural questions seem poised to become increasingly important for the relationship between data-curation and the overarching scientific (and publishing/dissemination) process.

1.6 Navigating the proliferation of research data

Whatever the motivations of scientists in curating research data, a further ineluctable detail of contemporary science and publishing is the large volume of (meritorious) work being produced. No doubt, it is better to have more good science than less, but the sheer scale of research work presents a challenge both to individual scientists (who are charged with making their contributions known to the relevant communities that can leverage them) and to technologies powering science as a whole.

Since it is impossible for any one person to read all scientific literature produced at any one point in time—or even all literature confined to a single specialty area, such as oncology, cardiac care, or COVID-19—scientists envision AI tools that could guide researchers toward relevant papers and data sets. More advanced search engines for scientific documents—if these can actually be implemented—would help investigators cut through the mass of literature and hone in on specific studies that are precedents or theoretical foundations for their own work. In short, better search tools could counteract the challenges posed by rapid expansion in the volume of scientific work. Scientists might then have the best of both worlds: a vibrant scholarly community that produces large quantities of credible science and, simultaneously, technologies to obviate information overload.

This is an encouraging idea, but there seems to be little evidence that search tools specifically designed for science perform noticeably

better than generic web searches. At best, truly accurate publication-search capabilities appear to remain a project for the future. A good case study in the current state and limitations of publishing technology is offered by the CORD-19 corpus, curated by the Allen Institute of Artificial Intelligence, to promote text and data mining targeted at literature related to SARS-CoV-2 and (to the degree that they may benefit COVID-19 research) coronavirus studies in general. The CORD-19 compilation provides machine-readable full-text versions of over 280,000 publications (as of mid-2021). We review the features and architecture of CORD-19 in Chapter 3. For the moment, we will simply point out that the data scientists who formulated CORD-19 openly acknowledge limitations in their methodology to obtain full-text representations and to curate them in a searchable manner. They even suggest the need for "a call to action," encouraging publishers to develop "distribution formats [for] scientific papers," which are less ambiguous than PDF (i.e., less opaque in text-encoding), to share publication texts in "structured format[s] like JSON, XML, or HTML," and to embrace "strict schemas" for article meta-data.[12] If the availability of machine-readable text serves as a qualification distinguishing which papers are included in aggregative corpora—and also which papers, once included, are more prominent in search results due to their being properly annotated (with demarcated keyphrases, findable data links, and so forth)—then publishers have an incentive to adopt paradigms akin to those which the Allen Institute is advocating in this context.

Similar challenges confront finding and integrating research data across disparate projects. It is difficult to search within data sets because there is no obvious foundation to look for key phrases, for instance, the way that search engines can match search terms against the raw text of a publication. There is, in general, no "raw text" within a data set that can be scanned for keywords and phrases. A related problem is that multiple data sets can be hard to aggregate together, even if they use similar methods applied to similar real-world problems. Unless there is a rigorous isomorphism between the statistical parameters and data-types employed between two kindred research projects, there is no automatic process to map one project's parameters onto another's so that they may be analyzed or visualized as a whole, or subjected to integrated statistical processing. Even subtle differences in parameters' variance, distributions, ranges, and data-acquisition methods complicate attempts at data aggregation spanning two or more research projects.

In light of these difficulties, scientists have proposed numerous formats for describing published data sets, with the hope that common representations would make data sets more searchable and interoperable. One aspect of these standardization projects is the notion of "microcitations" (viz., strategies to demarcate individual parts of a data set as citable references), analogous to citations of specific pages within a published document. The process of forming microcitation-targets in the context of specific data sets, however, depends on the format through which the data is encoded. Object values in JSON, SQL table rows or columns, XML document nodes, or record-tuples for formats such as **numpy** or CSV may all be feasible microcitation sites. Given that data sets might employ any of these representations (or many others), it is not unproblematic to standardize a general-purpose micro-citation format.

These data-sharing and text-mining challenges are significant, but they also give us a lens through which to anticipate what kinds of data curation and document-preparation technologies will become popular in the next phase of scientific publishing. It is reasonable to guess that scientists will benefit from standardized,

[12] *See* Lucy Lu Wang, et al., "CORD-19: The COVID-19 Open Research Dataset" (https://arxiv.org/pdf/2004.10706.pdf), page 8.

multi-disciplinary data representations that support micro-citations, that allow publication texts to cite specific parts of data sets (much as they cite other articles), and that allow code to be re-used across multiple research projects as a means to achieve data integration. We consider this a hypothesis as to the general priorities that will shape scientific computing and publishing technologies moving forward. The actual software engineering and data-modeling structures and design patterns that might realize these general goals will be the subject of much of our analyses in several later chapters, particularly Chapters 5, 6, and 9, and (in the specific bioimaging context) Chapters 7 and 8 as well.

1.7 Summary

Most of the chapters in this book will be focused on different approaches to data modeling, such as Type Theory, Conceptual Spaces, or Graph Database architectures. Our emphasis from a *theoretical* point of view will be on data-modeling representational paradigms, whose goals and criteria are oriented toward software engineering and the interoperation of distinct software components. That is to say, we advocate for data-representations whose rules and conventions prioritize the implementation of software components that produce, share, and consume the modeled data. As an underlying assumption: any database, data set, or information space should be engineered with the expectation that multiple (not fully isomorphic) software components will be interacting with that data; and that parts whereof will be passed and shared between such components, implying that data should be structured to facilitate cross-component communication.

We propose that most of the theoretical constructions introduced vis-à-vis hypergraph or code models may be concretely instantiated through virtual machines via which query-evaluation engines may be implemented. A full exposition of the design and construction of such virtual machines is outside the scope of this book, but we present several analyses here that serve as precursors to a more formal elucidation of hypergraph-query virtual-machines implementations. More broadly, we use the architecture of virtual machines within this category as an organizing motif for analyses conducted in Chapters 6 and 9.

From a more practical or "applied" point of view, we will call attention in particular to biomedical research projects that synthesize information with variegated disciplinary provenance and diverse data profiles. Of course, much biomedical research is inherently interdisciplinary. However, new breakthroughs and new research methods and technologies have accelerated the cross-disciplinary insights of research in several specific biomedical disciplines, yielding diagnostic, prognostic, and explanatory models that cut across biophysical scales (molecular, cellular, tissues, organs) and data-acquisition modalities (proteomics, genomics, biopsies, image processing, lab assays—such as for biologic sample analysis—and so forth). Examining literature where these integrative studies are described, it becomes clear that scientists often construct the software ecosystem powering their research in ad-hoc ways, piecing together diverse software components (sometimes standalone applications, sometimes code libraries, or some combination of the two) designed for specific disciplinary contexts.

We present arguments in later chapters to the effect that the relatively informal trial-and-error approach often taken to integrating multi-disciplinary biomedical data can act as an impediment to research replication and the systematic evaluation of interdisciplinary research findings. This is one reason for engaging in a detailed review of data profiles, data modeling paradigms, and data integration techniques, so as to lay the foundation for a software ecosystem that can support the emerging paradigm

of transparent and replicable research data and digital scientific resources.

We do not claim any special insights into interdisciplinary biomedical methods or data sharing as such; it is commonly acknowledged that data sharing is an increasingly important part of both research and clinical practice, and that breakthroughs in fields such as oncology and immunotherapy will depend on carefully calibrated multi-disciplinary data integration. However, while undoubtedly (in light of today's highly interconnected digital-health ecosystem) many biomedical and clinical data spaces are utilized by multiple (independent) software components, there are intricate design challenges that confront the engineering of data sources, which allow autonomous components to leverage their data in consistent (but flexible) ways. Our biomedical software ecosystem, we argue, still remains more fragmented and unsystematically designed than would be warranted given how profoundly different biomedical subdisciplines have been interconnected in recent years.

In particular, we examine circumstances through which interoperability impediments contribute to antipatterns we refer to as "ecosystem fragmentation," and (in Chapters 4, 7, and 8) describe what we call "multi-aspect modules" as potential corrective strategies. We use *multi-aspect* to qualify modular design patterns aiming for components intermediate in scale between domain-specific code libraries and monolithic scientific applications. Here the goal is to merge benefits of standalone applications (in particular, combining multiple software-engineering concerns, such as GUIs, data persistence, and data sharing/serialization, into a single code base) with those of smaller-scale code libraries (viz., interoperability, and the flexibility of mixing modules into standalone applications in different combinations, tailored to the needs of individual projects).

As for Dr. Rochelle Walensky's testimony whose controversy was discussed earlier, the CDC director might be justly held accountable for misrepresenting a specific COVID-19 study, but she can hardly be faulted for attempting to base her testimony on peer-reviewed scientific literature. The problem is that—although many people believe government policies should be grounded on scientific evidence and should respect scientific consensus wherever possible—all too often, there simply *isn't* scientific consensus, even in light of substantial real-world data. Such lack of consensus should not inhibit policymakers from basing government decision on data-driven, empirically minded deliberation, but it implies that scientists need a more sophisticated model of how to translate scientific findings into public policy insofar as the science itself is sometimes inconclusive and contradictory.

One way to achieve this, as explicated in this volume, is by formulating more sophisticated presentations of research data, of archives tracking multiple research projects, and of document-preparation in conjunction with data curation. Equally important are the software and algorithms used to integrate disparate data sources (while also modeling the anomalies and structural anisomorphisms that can make integrations inexact), ideally leveraging innovative ideas in query-engineering and data models that come to the fore within the bounds of a sufficiently multi-disciplinary perspective, some of which we hope to have advanced here.

PART I

Biomedical data formats and data integration

This page intentionally left blank

CHAPTER

2

Data structures associated with biomedical research

OUTLINE

2.1 Introduction

COVID-19, cancer, and cardiac care are among the most significant health-care challenges of our time. The devastation wrought by the SARS-CoV-2 pandemic, both in terms of lives lost and global economic impact, is well-known. Cancer and heart disease are even deadlier. COVID-19 seized the world's attention in early 2020; cancer and heart disease have not had their own single crisis moment, but they have been at the forefront of scientific attention for decades, spurring treatment innovations and scientific breakthroughs (a silver lining for all their tragic damage).

COVID-19 is a coronavirus very similar to the SARS strain that caused many deaths in 2003; like all viruses in their class, these pathogens use spike proteins to invade the human host's respiratory system, spreading to the lungs and some-

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00007-6

times affecting cardiac and neurological functioning as well. Many cases of COVID-19 are mild or asymptomatic, but for still-obscure reasons about 16% of infected individuals require hospitalization, and the disease has a 3–4% mortality rate. All of this is widely known, reported in everyday newspapers and web sites as well as scientific journals.

But how *do* we know these details? Different facets of COVID-19—its genetics, proteins, biomechanics, epidemiology, mutations, treatment options, and so forth—are the province of different biomedical specializations. The complete picture of COVID-19 therefore has to be assembled from many different lines of research. Much of our information about the virus was acquired soon after the start of the pandemic, and the development of effective vaccines has progressed at a torrid pace. Such accelerated results, unprecedented in medical history, reflect well on the current state of biomedical knowledge and the competence of practitioners of biomedical sciences worldwide. Although COVID-19 data is imperfect, and a lot remains mysterious about this disease, scientists' opinions should generally be taken seriously (and not politicized or obfuscated for ideological reasons).

Nevertheless, there is no harm in seeking to get a deeper understanding of *how* COVID-19 science has progressed so quickly. For anyone approaching COVID-19 from an angle outside of biology or health care proper—be that public policy, ethics, economics, sociology, or technology (and so on)—trying to understand the chain of scientific reasoning is not a matter of preempting scientific expertise, but rather just attempting to comprehend the COVID-19 crisis in a well-informed manner. What were the crucial insights gleaned from SARS-CoV-2 genomics, morphology, and biomechanics, or from contact tracing and clinical observations, which allowed a useful picture of COVID-19 to fit together in just a few months in 2020? How does the cryptic numeric data driving models of SARS-CoV-2's genes, proteins, or anatomy get translated to intuitive pictures of the virus as an active biological agent, with its specific infectious tendencies and deadly effects on human hosts? How do we observe the virus's effects on lungs and other organ system, on cognitive functioning, and on long-term health? How we do quantify patients' immunological response to the disease via antibodies? How do we detect viral fragments and particulates on surfaces or in the air, and simulate their scatter-patterns? How do we articulate mutations that have caused SARS-CoV-2 to evolve into distinct variants with divergent pathological profiles?

To be sure, a thorough recount of the scientific history unfolding through the COVID-19 pandemic would require a much longer book than this one. We will not, in other words, directly attempt to excavate scientists' detective work in early 2020 in the manner we just hypothesized, however much that is a story which inevitably *will* get told sometime after the pandemic has receded. But we *can* touch on one very specific field within this overarching trajectory, namely the role of computer software and the kind of data which forms a first step toward comprehensive models of diseases (and their treatments and interventions). This applies, of course, to cancer/oncology and cardiac care as much as to COVID-19.

This prefatory chapter will focus on certain specific data types which emerge from contemporary research methods and clinical/diagnostic data-acquisition modalities (including specialized equipment which itself represents a form of technological breakthrough, such as high-throughput gene sequencing). We will also consider how the goals of "personalized" medicine have motivated the search for data-integration methods working on these data structures emerging from new biomedical technologies. Our goal is to look at certain concrete data structures—their computational profiles, representation formats, and scientific foundations—as a precursor to later chapters' more theoretical discussions of data types as software artifacts.

"Biomedical Technology" is an umbrella term that encompasses many more specific topics and subdisciplines. Different investigative modalities—from genomics to proteomics to bioimaging (to name just a few)—tend to have their own technological ecosystem. Scientists who concentrate on specific fields within the biomedical umbrella therefore become familiar with technology (meaning both computer software and lab equipment) that is used in their area of study. There is less impetus for an overarching expertise that would encompass biomedical software in a more holistic sense, which potentially yields lacunae in biomedical software engineering. There is, in short, relatively little research into design patterns that might be generally applicable to biomedical software as a whole.

At the same time, with the rise of precision medicine, patient-centered care, and systems-biologic paradigms which integrate multiple anatomical and biochemical scales (from molecules to cells to tissue and organ systems), biomedical research has become interdisciplinary to an unprecedented degree. One consequence of this (generally positive) trend is that software ecosystems and computational paradigms targeted at specific subdisciplines prove to be suboptimally fragmented. Cross-disciplinary research implies the need to connect hitherto isolated software ecosystems together.

Our goal in this chapter is to make these issues tangible by identifying several specific biomedical subdisciplines and the data structures they tend to generate. These are the kinds of heterogeneous data models that need to be integrated to provide a computational foundation for multi-disciplinary biomedical research. We are not attempting to develop a comprehensive list of bioinformatic data-types, but merely to describe certain data models that are especially important and recurring in contemporary science. Hopefully these concrete examples can help to orient our analyses when we transition to more abstract discussions about data types, data models, and software-development paradigms.

This chapter will consider "personalized medicine" in the context of COVID-19 (later chapters will address personalized medicine in its more traditional settings, particularly immunotherapies as promising approaches to cancer care), so we adopt COVID-19 as a lens for examining the origin and clinical significance of data structures emerging from technologies such as Next-Generation Sequencing and bioimage analysis via Computer Vision. However, data structures characteristic of these technologies are, for the most part, widely used in clinical and diagnostic practice; they are not specific to COVID-19 (here we will not emphasize research that is more narrowly targeted at SARS-CoV-2, such as biophysical analyses of the virus's proteins, or epidemiological studies of the pandemic's origins).

Gauging public sentiment, it has been suggested that patients during the COVID-19 pandemic are increasingly reluctant to participate in conventional trials, wherein they run the risk of being assigned to a control group where they might lose the opportunity to receive life-saving treatment.[1] In this context, the approach of Shrestha *et al.* [32] (say, as elaborated below) may be justified as more consistent with patients' wishes for their own care, wishes that should be taken seriously in patient-centered contexts.

Shrestha *et al.* represents only one publication, but their philosophy of trial design perhaps portends a paradigm-shift away from individual, controlled studies and toward nonrandomized, concurrent trials. In lieu of single investigations, comparing an experimental group with a control group, concurrent trials allow

[1] *See*, for instance, [45], or https://www.pharmacytoday.org/drugs/drugs-2020-04-22-story4. Also [4] is an interesting discussion about the merits of double-blind trials (and in particular the idea of "unblinding" follow-up studies), albeit well before COVID.

the results of different studies to be compared against one another; each trial serves as a *de facto* control vis-à-vis each concurrent trial. Integrative data-modeling tools thereby steers patients into different trials based on personalized data packages (which may include immunological, cardiovascular, neurological, and other general medical-history data) obtained for each patient prior to their involvement in a trial, and likewise made available for subsequent analysis.

2.2 Personalized medicine in the context of COVID-19

Within just a few months after COVID-19 became a global pandemic, doctors and researchers were developing and experimenting with a diverse range of treatments and remedies against SARS-CoV-2. The treatments that were in some sense tested or adopted include monoclonal antibodies, remdesivir, dexamethasone and other steroids, anti-inflamatory drugs (such as tocilizumab and sarilumab), convalescent plasma, baricitinib (a rheumatoid arthritis drug for which COVID-19 is an off-label use), and anti-malarial drugs (specifically chloroquine and hydroxychloroquine).[2] These treatments, that is to say, were either approved (albeit provisionally or "for emergency use") by the United States FDA and other government agencies, pending clinical trials, or reported anecdotally to be helpful in combating COVID-19. In either case, they were relatively widely adopted in clinical settings. More rigorous attempts to validate claims of any treatments' effectiveness, however, have proven inconclusive (at least as of mid-2021)—certainly no pharmaceutical or clinical intervention has demonstrated success rates comparable to the first wave of vaccines that were approved toward the end of 2020. In short, every well-studied COVID-19 intervention (other than vaccination) appears to be successful in some patients and not others. These mixed results generate questions of their own: what are the biologic mechanisms that cause different treatments to play out differently in distinct patient populations?

Of course, it is possible that associations between treatments and outcomes—at least in the SARS-CoV-2 context—is driven mostly by random chance, or that treatments are correlated to, rather than causative of, outcomes. The COVID-19 mortality rate among hospitalized patients appears on average to be less than 25% (even though over four million people worldwide have died from the pandemic, as of summer 2021). Accordingly, many people even with serious cases of COVID-19 will recover just via statistical distribution. Though it is possible that treatments received in hospitals account for some of the recovery rate—the mortality rate amongst hospitalized patients has diminished over time, which implies that hospitals have become more skilled in caring for COVID-19 patients and that treatment plans have become more refined—many patients hospitalized for the disease might well have recovered anyhow (even without additional pharmaceutical or clinical interventions). Given this interpretive uncertainty, it has been difficult for researchers to definitively show that particular treatments do in fact improve recipients' chances of recovering from COVID-19. Nevertheless, both observational evidence and clinical trials suggest that a number of COVID-19 treatments do have some positive benefits, even if not for all patients.

This situation then leads to the question of why particular treatments benefit some patients more than others. Such considerations are of scientific interest, of course—clarifying how drugs or other interventions interact with the SARS-CoV-2 virus enhances our knowledge about its infectious mechanisms and the progression of COVID-19—but more immediately it is of clinical interest, because doctors need guidelines

[2] *See* https://www.mayoclinic.org/diseases-conditions/coronavirus/expert-answers/coronavirus-drugs/faq-20485627.

on when to administer which treatment(s) to which individual patients. Since no COVID-19 remedy (apart from vaccination) clearly outperforms others under most circumstances, scientists have attempted to study COVID-19 treatments with the goal of giving doctors more detailed information to work with when making these sorts of clinical decisions. The overarching project of a significant subset of COVID-19 research, in short, has been to establish criteria allowing doctors to predict which treatments are most likely to succeed for individual patients, given details of their immunological profile and prior medical history.

In this sense, the goals of COVID-19 research overlap with methodologies that have been established in the context of other research areas—notably cancer and AIDS—under the general rubric of "precision medicine." As a field of research, precision medicine is focused on the correlation between patient-specific biomedical details and the likelihood that particular interventions will have positive effects for particular patients. In contrast to conventional clinical trials, which consider patient-specific details only at a rather coarse level—merely observing obvious data-points such as age, gender, and ethnicity—precision-medicine research attempts to curate a much more detailed picture of patients' medical and sociodemographic profiles, either as part of formal trials or as observational details in a clinical setting. Ideally, precision medicine is motivated by the goal of analyzing patient-profile-to-outcomes correlations not only retroactively (detecting statistical patterns in prior outcomes) but also prognostically. In effect, once a particular sort of treatment has been identified as especially favorable for patients with certain characteristics, it is reasonable to project that future patients with similar characteristics should be given similar treatments. This intuitive and obvious point, however, masks empirical and practical difficulties; in particular, it is not obvious how to quantify the notion that current patients are "similar" to prior ones. Indeed, one of the provisional results of precision-medicine research thus far has been to highlight how statistically significant points of resemblance between patient profiles do not necessarily line up with how we *intuitively* group patients together by visible factors, such as age, race, or gender.

In sum, precision medicine has been guided by the thesis that compiling detailed patient-centered information can advance clinical medicine by identifying which treatment options are more likely to succeed for individual patients. In the context of COVID-19, both the diversity of legitimate clinical interventions and the failure of any one treatment to show unambiguous success for a broad spectrum of patients point to the potential usefulness of patient-centered approaches. If doctors have detailed information about COVID-19 patients' immunological profiles and clinical history they can—at least in theory—make informed decisions about which treatment options have a higher probability of success. Such predictions would not be a matter of guesswork; instead, statistical analysis of COVID-19 cases in the past, preferably backed by machine learning, would detect correlations between patient data and recommended treatments. In effect, the spectrum of possible COVID-19 interventions (antibodies, steroids, convalescent plasma, and so forth) serves as a natural classifier, grouping patients into clusters based on pre-treatment profiles indicating that one or another COVID-19 intervention has, with some probability, a good chance of helping the patient. Researchers have therefore been seeking empirical clues for how to classify patients into categories based on recommended treatment plans.

We will examine in this chapter how the goals of precision medicine have spurred scientists to propose more sophisticated data integration and data management tools in the context of clinical trials and biomedical laboratory investigations. Aside from opening new avenues for empirical research, the *goals* of precision medicine have spurred new ways of thinking about the compu-

tational ecosystem which supports biomedical research and clinical practice.

2.2.1 Precision medicine as a catalyst for biomedical data sharing

Precision medicine is based on the scientific realization that the effectiveness of a certain clinical intervention—for instance, immunotherapy as a cancer treatment—is dependent on factors that vary significantly between and among patients. In the case of immunotherapy for cancer, assessing the likelihood of favorable outcomes requires genetic, serological, and oncological tests which need to be administered prior to the commencement of treatment. Therefore analysis of precision medicine outcomes requires detailed immunoprofiling—building immunological profiles of each patient before (and perhaps during/after) the immunotherapy regimen—alongside an evaluation of how well the patient responds to the treatment, with the best outcome being cancer remission or non-progression.

The contemporary emergence of precision medicine is driven, in part, by advances in diagnostic equipment which enable immunological profiles to be much more fine-grained than in previous decades. This level of patient-profiling detail enables granular three-way analysis involving (1) patients' immunology; (2) treatment regimens; and (3) clinical outcomes. This three-way approach has the goal of identifying signals in immunological profiles that indicate which therapeutic interventions are most likely to engender favorable outcomes. As with any statistical analysis, predictive accuracy increases in proportion to the amount of data available. As such, further refinement of precision medicine depends in part on data aggregation: pooling a number of observational studies wherein patients' immunological profiles are described, in detail, alongside reports on treatment plans and patient outcomes.

Within the scientific research community, organizations such as the Society for Immunotherapy of Cancer (**SITC**) and the Parker Institute for Cancer Immunotherapy (**PICI**) have developed programs and tools to share immunotherapy research data, which join older (but less cutting-edge) projects, such as Cancer Commons or the **RSNA** (Radiological Society of North America) Image Share program. In the COVID-19 population, we are similarly presented with multi-dimensional patient data. Such information traverses molecular testing to identify the virus's genetic material or the unique markers of the pathogen itself; antigen testing (albeit less accurate) to identify specific proteins found on the outer surface of the virus; mapping the genomes of SARS-CoV-2 to learn how it mutates; analyzing blood samples for the presence/absence of antibodies in response to a prior SARS-CoV-2 infection; using high-dimensional flow cytometry to perform taxonomic breakdown of patients into distinct immunotypes related to disease severity and clinical parameters; profiling patients' immunological state prior to the start of treatment and quantifying patients' immunological response once treatment has begun; calculating plasma viscosity (**PV**) for detection of unusually high levels of fibrinogen—leading to atypical blood clots (that are refractory to standard anticoagulant therapy); monitoring cognitive, neurological, or cardiovascular symptoms in "long haulers," and so forth.

One area where data integration is significant for COVID-19 research is that of protein biomechanics. For instance, [47] identifies 64 different proteins that are therapeutically relevant to COVID-19. Each of these proteins can be connected to molecular data, bioassay information and, in many cases, to clinical trials that test therapies wherein the specific protein acts as a target for inhibiting SARS-CoV-2. In combination, aggregating all of this information yields a heterogeneous (but interconnected) data space, characterized by data derived from multiple scientific disciplines and multiple laboratory methods.

Because biomedical diagnostic and investigative laboratories need to use highly specialized software, raw laboratory data is often excluded from data-sharing initiatives. Instead, the information which is shared between hospitals or research centers tends to be a simplified overview—adhering to standards curated by groups such as **OMOP** (Observational Medical Outcomes Partnership) Common Data Model or **CDISC** (Clinical Data Interchange Standards Consortium)—skews more toward succinct summaries of diagnoses, treatments, and/or outcomes. Limitations in sharing more granular diagnostic or prognostic data have been identified as obstacles diminishing the value of data-sharing initiatives. As one example, in a paper discussing research into the sequencing of immunoglobulin repertoires (Ig-seq), [12] comments:

> A major challenge when performing Ig-seq is the production of accurate and high-quality datasets [because] the conversion of mRNA ... into antibody sequencing libraries relies on a number of reagents and amplification steps ... which potentially introduce errors and bias [that] could alter quantitation of critical repertoire features. ... One way to address this is by implementing synthetic control standards, for which the sequence and abundance is known prior to sequencing, thus providing a means to assess quality and accuracy.

The underlying problem in this context is how different laboratories may use different techniques and protocols to achieve similar diagnostic/investigative goals (*see also* [17], [19], [37], [46], etc.). Consequently, when data is merged from multiple hospitals, it is likely that the raw data derives from different labs, which can lead to situations where protocol variations across each site can introduce errors and bias that contaminate the aggregate data-sharing results. (We should note that Ig-seq and related "repertoire sequencing" is quite relevant to coronavirus immunology because these techniques help quantify patients' immune response to infection, which is consequential both for explaining the differences between mild and severe cases and observing the effectiveness of interventions such as vaccinations or antibody treatments; *see* [10], [21], [30], [11], [13], [41], [6], [29], [14], etc.)

In the context of COVID-19, Shrestha *et al.* argue for "precision-guided studies" to be prioritized "[r]ather than conducting trials using the conventional trial designs and poor patient selection" (page 1). To accomplish this, the authors recommend "large multicenter trials" which incorporate "predictive enrichment strategies ... to identify and thus target specific phenotypes [patient-profile characteristics], potentially raising the possibility of positive trial outcomes." The underlying problem identified by Shrestha *et al.* is acquiring sufficiently large trial cohorts in contexts where trials are to be targeted at fine-grained patient populations with specific pre-treatment immunological profiles. This problem can be ameliorated by merging prospective patients from multiple institutions, such that concurrent trials spanning multiple healthcare settings can be launched as part of a comprehensive approach to comparing treatment options. Such large multicenter trials can produce "large cohorts of patients in a shorter time period" while also steering patients toward more favorable treatment courses.

In short, Shrestha *et al.* explicitly challenge the conventional wisdom, which assigns "gold standard" status only to *randomized* trials, arguing that the benefits of larger trial sizes and quicker trial initiations outweigh whatever statistical value is compromised by earmarking patients into trials based on educated guesses as to favorable outcomes (which is warranted by patient-care ethics in any case). The authors also argue for a "robust data infrastructure" that would combine the efforts of clinicians, researchers, and data scientists. In effect, aggressive data-curation would substitute for double-blind trial design as a means of ensuring the scientific value of trial outcomes.

2.2.2 Software alignment for COVID phylogeny studies

The study of SARS-CoV-2 mutations is another area where software alignment can prove to be important. Analyses in 2020 suggested that variations in the viral strain causing COVID-19 symptoms may be partly responsible for divergent immunological responses to the virus across the patient population [31]. If one patient responds either less favorably or more favorably than the average patient-response to a given treatment, clinicians need to assess whether this difference can be explained solely by the patient's prior immunological profile, or whether the patient has been exposed to a genetically divergent viral strain. In 2021, of course, predictions about SARS-CoV-2 mutations came to pass, with at least four variants appearing to be sufficiently dangerous (by virtue of their infectiousness and/or lessened effectiveness of treatments or vaccines) to undo progress which has been made against COVID-19 in many parts of the world.

Comprehensive models of COVID-19 variants require a combination of genetic, proteomic, anatomic, and epidemiological information, because mutations can only take hold if they modify the virus's morphology and/or infectiousness in ways that are conducive to that strain replicating. The **delta** strain, for example, which (as of mid-2021) has been the most wide-spread mutation [35], carries a genetic variation (designated as "**D614G**") which results in a denser array of spike proteins, thereby reinforcing the biomechanic pathway, which the virus uses to infect the host's respiratory system [16], [23], [27], [48], [3], [39], [49], etc.

Modeling SARS-CoV-2 evolution across the globe is a massive project. There have been over 200 million COVID-19 cases worldwide (as of mid-2021), in virtually every nation on earth, so a complete phylogenetic picture of SARS-CoV-2 in humans would need to pool data from many different healthcare systems. Yet, even technically detailed analysis of the phylogeny of SARS-CoV-2, such as that conducted by Dearlove *et al.* (as reported at the end of 2020) only considered 27,977 patients (about 0.1% of global cases), with almost half from the United Kingdom [8]. Hence, achieving something resembling a holistic picture of SARS-CoV-2 mutations and how they might affect clinical treatments, would require many parallel studies analogous to that of Dearlove *et al.*

This then raises questions of study alignment: calculating viral phylogeny requires making technical decisions about how genetic sequences should be acquired and analyzed, decisions which may vary among research teams. For example, Dearlove *et al.* describe several computational steps they had performed both to normalize each SARS-CoV-2 genome sequence in their data set for cross-comparison and to run predictive simulations (used to estimate whether divergence between sequence-pairs are the result of localized, random mutations or, conversely, an indication that SARS-CoV-2 is evolving into further distinct strains). Clustering SARS-CoV-2 genomes into variants, that is, identifying which mutations are random and which appear to be propagating to subsequent viral generations, involves making computational and biological assumptions, such as how to statistically marshal genomic data so as to quantify the prevalence of a mutation, and how to estimate whether a particular mutation confers an adaptive benefit to the viral agent (e.g., an ability to elude antibodies targeting structural proteins).

Given that modeling viral phylogeny requires certain computational assumptions and biological guesswork, data from multiple studies can only be reliably integrated if there is some degree of alignment across their methodology. As such, research teams should document their protocols in a manner that permits assessment as to whether protocol differences might compromise the resulting data. One way to achieve this is to model the protocol itself as a data type in

a general-purpose programming language (such as C++, for the sake of argument). For each study, such as Dearlove *et al.* cited above, there would then be a C++ object encapsulating details of the researchers' protocols and computational workflows. Protocol-alignment would in this context be one part of a common framework to quantify the epidemiological significance of SARS-CoV-2 mutations.

In short, a holistic global picture of SARS-CoV-2 must represent SARS-CoV-2 mutations which have been deemed phylogenically and/or clinically significant (i.e., having potential either to influence the overall evolution of COVID-19 and/or to have some bearing on clinical treatments), and must *also* represent divergent SARS-CoV-2 strains carrying those mutations. These data-points are then the basis of further details such as the following: When did a given strain and/or mutation first appear? Is the strain/mutation geographically localized? What is the proportion of different strains/mutations in a geographic area? Is there evidence that different strains/mutations affect a patient's immunological response to COVID-19 and/or the effectiveness of vaccines, antibody regimens, steroids, or other clinical interventions? How can genetic mutations within the SARS-CoV-2 virus be correlated with structures in the spike proteins encoded by the viral genes? This last question points to the importance of integrating genomic data with 3D molecular models (*see* [44], [7], [20], [2], [28], [26], [1], for example). Whereas data structures modeling the viral genome are composed of nucleotides—and, at a higher scale, Open Read Frames (ORFs)—data structures describing the biophysics of glycoproteins involve protein architecture and chemical bonds [9], [22], [25], [33], [24], [36], etc. Analyzing how SARS-CoV-2 genes affect the production of glycoproteins therefore requires annotating and cross-referencing nucleotide/ORF data structures with 3D molecular models encoded in formats such as MOL or Protein Data Bank (PDB) [43], [5], [42], [38], [15], [40], etc.

Object-Oriented models for genomic phylogeny analysis have been presented in [34] (Java) and [50] (C++), although these do not provide object models to extend from genetic information toward clinical, proteomic, or imaging data. We are not aware of Object-Oriented models proposed as representational devices for COVID phylogeny in this more holistic and interdisciplinary modality, but this form of software design would be consistent with code developed in contexts such as oncology, for example, the computational simulation of tumor growth, which we will discuss in Chapter 4. COVID Phylogeny Object Models could serve as a nexus for merging temporal and geographical data concerning the epidemiology of SARS-CoV-2 mutations with genomic data demonstrating which mutations are significant, as well as clinical data tracking correlations between mutations and treatment outcomes. Supporting multi-trial data integration would therefore also introduce new requirements for clinical trial software, which we discuss further in the next chapter.

2.2.3 Personalized medicine and immuno-profiling

Though some of the data related to immunological profiles may be sociodemographic or part of a patient's medical history (fitting nicely within conventional Clinical Research Network models), contemporary immunoprofiling is powered by highly specialized diagnostic equipment and methods, which require special-purpose file formats and software. For instance, one dimension of immunological profiling is "immune repertoire"; the more robust a person's repertoire, the wider variety of antibodies they can produce to fight off pathogens. Immune repertoire is often measured by studying genetic diversity in B-cells; in recent years, this has been done using "Next-Generation Sequencing" (NGS), which produces files in formats such as FASTQ. Another dimension of immunological

Immunological Profile Dimension	Lab/Clinical Method	File Format
Sociodemographic	Scan patient records	CSV, SQL
Medical history	Scan patient records	CDISC, OMOP, PCOR
Immune repertoire	NGS	FASTA, FASTQ
Cell-type classification	Flow/mass cytometry	FCS
Immune/tumor microenvironment	Confocal microscopy	DICOM, OME-TIFF, CoCo

FIGURE 2.1 Table outlining several data formats and the contexts where they are acquired.

profiling is quantifying the proportions of different sorts of blood cells in a patient's blood sample, which is typically done via flow cytometry or mass cytometry, yielding FCS (Flow Cytometry Standard) files. The immunological evaluations which are the goal of these methods are sometimes called cell-type classification or "Automated Cell-Type Discovery and Classification" (ACDC).

As outlined in the table (Fig. 2.1), immunological profiles draw on a diversity of data formats and lab/data-acquisition modalities (this table is not intended to be a complete list of criteria or file formats, but rather to indicate the range of diagnostic technologies and data formats that are relevant to immunoprofiling). This table hopefully indicates how immuno-oncology data sharing is inherently more complex when compared to data-sharing initiatives that focus primarily on clinical outcomes—such as OMOP, CDISC, or the Patient-Centered Outcomes Research Network (PCORNET).

Formats such as the OMOP Common Data Model (CDM), the PCORNET CDM, or the CDISC specifications promote data sharing primarily through SQL-style tables, where data analysis and extraction can be achieved via conventional SQL queries. The situation is very different, however, when the data that must be exchanged derives from specialized hardware and software, which demands special-purpose file formats, parsers, and query engines. This problem-space is accentuated when preparing multi-site sharing that can span dozens of hospitals, research centers, and/or laboratories. Problems of cross-institutional data integration will be analyzed further in the next chapter.

Prior to that discussion, the following section will examine in detail the file formats and data structures, which were briefly mentioned thus far in this chapter. Our goal in this discussion is to document the kinds of data that are endemic to different branches of biomedical research. A separate analysis—one which to some extent depends on first describing the data formats involved—concerns how to fuse multiple data formats into a common overarching format, such as a "Common Data Model of Everything." One recurring theme we will encounter in the subsequent discussion is the goal of merging certain data profiles into others (e.g., FCS into DICOM), or else adopting common interchange formats (such as XML) in lieu of domain-specific binary formats that demand special-purpose parsers. We will examine both the strengths and weaknesses of proposals for adopting common formats rather than idiosyncratic "legacy" formats that are tied to particular laboratory methodologies for historical reasons. However, our main purpose in the current discussion is to establish basic facts about data formats in current use. Later chapters will analyze problems connected to the integration of these formats into common data models.

Note also that the following inventory of biomedical data formats is by no means exclu-

sive. In particular, we have largely neglected antigen tests, biochemical assays, and many other lab techniques that rely on chemical reactions to obtain lab/diagnostic findings. Our discussion here is oriented more toward methodologies that require relatively complex intermediate computational processing to arrive at clinically useful findings.

2.3 A review of certain commonly used biomedical data formats

Because a major focus of this book is the goal of *integrating* disparate data structures into a common format, it is an important preliminary step to examine the data formats that need integration to begin with. For reasons explained above, this list is by no means exhaustive or comprehensive; nevertheless, the data profiles considered here constitute a representative spectrum of structures which are important to contemporary bioinformatics.

Note in addition, that some of these data formats are demonstrated in computer code which we have prepared to accompany this book. The authors have republished and/or modified open-source libraries used to parse data in the formats discussed in this section, and we provide examples to demonstrate how these file formats are used in real-world contexts. For each format there is an accompanying code project (associated with a QT **.pro** file) which can be opened in the QT Creator IDE and compiled. The resulting executable will load a sample file and use parsers endemic to the file format being discussed. The details of each format may accordingly be studied by looking at the computer code for parsing the files and/or by running the demo programs through a debugger. More information about how to use these demonstration code libraries for pedagogical purposes is provided along with the code samples themselves.

2.3.1 DICOM (digital imaging and communications in medicine)

The DICOM format is closely associated with the PACS (Picture Archiving and Communication System) standard for sharing biomedical images; both formats together constitute the *de facto* standard for bioimaging in clinical and diagnostic settings. A PACS workstation is guaranteed to have certain capabilities with respect to the diversity of image-formats the software can receive and process from outside sources. This level of standardization is important because biomedical images are commonly shared amongst disparate institutions; for example, radiographic images generated in a hospital may be sent to a laboratory for diagnostic processing. The DICOM and PACS standards guarantee that recipients of images will be able to view and share them properly, assuming that the images are provided in certain canonical formats and with certain basic metadata (it is worth noting that, in the absence of metadata, it may be impossible for software to display images because of ambiguities in color depth, color format, image dimensions, and other details that determine how binary image data is translated to visual pixel renderings). These standards eliminate the sorts of delays that might otherwise be caused when a lab, for example, is unable to view images sent by a hospital or doctor's office.

Although the primary purpose of DICOM is to facilitate image transfer, a close second in importance is properly associating the images with relevant clinical metadata. Images are typically grouped into series, which are associated with specific patients and specific diagnostic goals (often diagnostic codes described via fixed vocabularies, such as the "Radiographic Lexicon," also known as RadLex). In the COVID-19 context, for example, lung x-rays may be requested to support a SARS-CoV-2 diagnosis and/or to test how far the disease has progressed into the patient's lungs. At a minimum, then, all images generated for that diagnostic purpose need to

be grouped together in the current clinical context (specifically that this image-series results from a possible COVID-19 diagnosis and has a specific clinical/diagnostic purpose) and associated with a specific patient (usually identified by some sort of anonymizing code). This residual non-image information needs to be tracked via metadata, typically represented within DICOM "headers."

Given that DICOM thereby serves as a forum for exchanging general patient information, not only images themselves, the DICOM standard was designed to be flexible and extensible. This has led the scope of DICOM to expand of the years, encompassing a greater diversity of graphics data (3D images, audio, video, ultrasound, etc.) and also non-image content (such as recommended treatment plans, image annotations, or diagnostic comments). The DICOM system is oriented to enabling these scope-extensions without breaking backward compatibility with older DICOM standards.

All DICOM data is associated with classificatory "tags," which indicate the type of information asserted by a given data-point and how it is encoded. Each tag is assigned a "group number" and an "element number"; this two-layer organization allows tags to be grouped together that serve a similar purpose or derive from a similar clinical or diagnostic context (the pairing of group numbers and element numbers may be roughly compared to XML namespaces and element names). Different DICOM group numbers are used to group tags into those specific to, for example, image data (e.g., image dimensions and color model), series/study data (date, patient ID, accession number, etc.), patient-specific information (birth date, gender, native language, and many other data-points), and numerous additional group numbers specific to different kinds of images (or other graphics content, such as video), which may be present in a DICOM series. Officially recognized DICOM groups are each assigned *even* group numbers. The *odd* DICOM group numbers can be used for non-standard DICOM extensions, but DICOM clients are not required to parse or recognize tags with odd group numbers. In effect, organizations are free to use odd DICOM groups to encode any information they believe is relevant, but there is no guarantee that third parties accessing their data will be able to interpret these non-standard codes.

Another important detail pertinent to the DICOM data model is that DICOM files freely combine binary and textual data. Most data in DICOM tags such as those just described is textual data, encoding character-based details, such as a patient's name or date of birth. Of course, DICOM also has to encode binary image data as well.

In this context, note that many "file formats" are actually *zipped* files that are unpacked into *folders* behind-the-scenes (so that they are actually "folder formats").[3] Programmers familiar with formats based on zipped folders might instinctively feel that the natural way to fuse textual and binary content is to create folders wherein the two sorts of content are spread across different individual files. For example, an XML file could encode textual data, such as that present in most DICOM tag-groups, whereas a series of image files could store the individual series images.

However, the DICOM standard embraces a much different architecture, where DICOM tags introduce data streams that may be either textual or binary; the tag itself (in fields immediately following the group and element numbers) asserts the number of bytes needed to encode information belonging to that tag, and those bytes may be interpreted either as character-streams or as binary sequences holding (in particular) image data. Whereas file formats that expand to unzipped folders are "file-based" (in

[3]For example, **e-book** files are merely zipped files encoding entire book directories, which generally include different files for individual chapters, images and other non-textual content, and metadata in formats such as XML.

that, once the unzip step is completed, information is spread across multiple files), DICOM is "stream-based": all DICOM data, from an identifier tag, which requires only one or two bytes to be encoded to a video asset that may run into megabytes of size, is inserted into a sequence of tagged byte-streams, which collectively span a DICOM file.[4]

The mixture of binary and textual data in DICOM tags—together with idiosyncratic features such as numeric group/element tag identifiers rather than character-based entity/field names—sets DICOM apart from most serialization formats, which makes it difficult to contemplate re-encoding DICOM data in other systems, such as XML. As a result, XML processing algorithms or Semantic Web query formats cannot be readily applied to DICOM data; instead, any application wanting to query or examine DICOM data streams needs to use DICOM-specific code libraries, such as DCMTK (this library is reproduced in the code accompanying the present book). DICOM's opacity to modern query languages has sometimes been a source of frustration for programmers, and has led to the development of proposals such as "Semantic DICOM," which involve extracting different sorts of data from DICOM series and warehousing this extracted information in databases that are amenable to analytic techniques which are not DICOM-specific. In so-called "semantic" DICOM, textual header data is placed in NoSQL databases of various sorts, whereas image data is marshaled into individual files, or stashed in an image database. The rationale for this "deconstruction" of DICOM data is both to analyze DICOM data using non-DICOM-specific algorithms and to merge DICOM information with data having other profiles. We will examine semantic DICOM proposals in greater detail in a later chapter.

[4] As an aside, DICOM tag-sequences can be nested in other tags, so technically DICOM files are hierarchical by analogy to XML or JSON; however, this hierarchical structure is much less common in the DICOM context than in other encoding schemes. It is reasonable to consider the DICOM format as first and foremost a string of sibling tags, rather than as a hierarchical document structure, for most purposes.

2.3.2 Next generation sequencing and other genomics formats

At the core of all genomics data is the four "letters" (ACTG, or adenine, cytosine, guanine, and thymine), which are the building-blocks of DNA and RNA (for RNA, thymine is replaced by uracil, marked with a letter U). However, genetic data is much more complex than merely strings of letters drawn from four choices, because genomics files also represent groupings of these elements (and also, in some contexts, points of divergence amongst two or more genetic sequences). Genomics data structures differ in how these groupings and comparisons are described.

An early and simple example of a genomics file type is FASTA (this name derives from the term "FAST-All," and originates in a software program that extended an earlier "FAST-Protein" application). The FASTA format is used for encoding amino acid sequences as well as nucleotides, so numerous letters may be used in addition to ACTG/ACUG. Each letter represents either a single nucleotide or a single amino acid, and sequences are represented by strings of the corresponding letters.

The other structuring elements of FASTA files are "comment lines," which precede sequence notations and provide context explaining the source of the sequences. Though in theory the comments are human-readable explanations, which are not formally part of the FASTA format, in practice certain structural norms have been adopted in the construction of FASTA comment lines, so that these conventions serve as a *de facto* extension to the FASTA standard. In particular, FASTA comments often employ references to databases from which sequences are obtained, using the "pipe" (i.e., vertical-line) character (|, Latin-1 code 124) as a field-

delimiter. A more recent variation on FASTA, called FASTQ, supplements this sort of contextual information with "quality scores" that measure the probability that a sequence has been read accurately by the physical equipment used for the gene sequencing operations that yielded the data being presented.

Genomics file formats have evolved alongside gene-sequencing technology. Genetics has been revolutionized in particular by "Next-Generation Sequencing," where many small DNA sequences are extracted in parallel and then assembled together with the aid of computer software. NGS has made it cost-effective to perform gene sequencing on a much larger scale (e.g., a human genome can be extracted in a single day), which has greatly expanded the quantity of genomics data available to research, while also generating a larger quantity of intermediate data (because NGS requires a computational infrastructure to derive complete results).

Both of these trends have affected the software ecosystem supporting genomics; a window onto this evolution can be gleaned by considering new file formats that have emerged to encode NGS-related data. For example, the Sequence Alignment Map (SAM) and Variant Call Format (VCF) are employed to notate both alignment/commonalities among gene sequences and their points of divergence.

In the case of VCF, data structures do not represent nucleotide or amino acid sequences directly, but instead notate the variation present in one sequence when compared against a reference sequence (such as the transposition, deletion, or insertion of individual nucleotides). The VCF format is column-based, where one column marks a position in the respective sequences where a variation occurs (i.e., the sequences have different letters, or one sequence has no corresponding letter at all) and other columns provide contextual information, including the type of variation and the origin of the divergent sample. The columns present in a given VCF file are labeled by a line preceded by a single "pound" character (#, or Latin-1 code 35), immediately before the raw data. Prior to this information are "header" lines that begin with a double-# and provide information *about* the column fields, including their identification codes and data types (such as strings, booleans, or floating-point numbers).

The SAM format is also primarily column-based, with an "alignment" section comprised of individual lines encoding 11 canonical fields as well as potentially other optional fields. The 11 mandatory fields correspond to technical details concerning how alignments between distinct sequences are discerned (either the number 0 or the asterisk character is used to represent missing values). One of these fields, treated as a single integer, is actually the binary encoding of a bitset; in other words, a string of boolean values, so the corresponding column actually represents a collection of roughly 12 individual true/false data-points. Optional fields, after the 11 canonical columns, are grouped according to two-character tags, accompanied by single-character encodings of types (there are only a few recognized types in this context, such as signed integers or single-precision floating-point decimals). In addition to alignment data, each SAM file has header lines, marked by an initial "at" character (@, Latin-1 code 64) and a two-letter code indicating the header record type. Each header line then has a sequence of tab-delimited fields, which in turn have their own two-letter identifiers. Note that the vocabulary for fields across the SAM format is fixed: every tag or field descriptor has (only) a two-letter code, and the set of codes recognized in each context is defined by the SAM standard.

When notating alignments, SAM identifies a *reference* sequence against which other sequences may be compared (i.e., aligned). In this context, SAM distinguishes between *padded* and *unpadded* reference sequences. The explanation for this distinction derives from the nature of sequence variation—one way that a variant se-

quence may differ from a reference sequence is in the *insertion* of a nucleotide or amino acid in a position where no corresponding element exists in the reference sequence. In these situations, the most straightforward way to notate the variation is to mark a gap in the *reference* sequence (typically via an asterisk). The potential problem with this notation is that it conceptually distorts the relation between the reference and the variation: the reference constitutes a basis against which a variation (or multiple variations) may be defined, so it is counter-intuitive to *modify* the reference-sequence in light of how variations diverge from it. A further complication is that modifying the reference propagates to modifying all variant sequences which do *not* have an insertion at the relevant position. In any case, some applications call for the use of "unpadded" reference sequences, which have no modifications (forcing insertions to be notated with more complex representations), whereas others are facilitated by "padding" the reference base with gaps where one or another variant possesses an inserted letter.

The preceding overview of genomic file formats has only considered structures that encode raw sequence data and its alignments or variants; this discussion has overlooked any of the tools or notations that build a coherent narrative on the basis of sequencing, such as observing how a genomic profile has emerged over time. For example, data visualization tools may be used to study genetic variations—including the mutations that have led to divergent strains of SARS-CoV-2—by picturing them as trees, where branches separate at the point of variation on a specific genomic site. In SARS-CoV-2, for instance, one can model the temporal evolution of viral variants via data structures identifying when a particular strain emerged, from which prior strain, and the genomic site where a mutation forms the basis of the variant's specificity. These data structures, however, are different in form and visualizations than the underlying genomic data itself.

2.3.3 The flow cytometry standard (FCS) file format

Flow cytometry uses a combination of lasers and fluorochromes to investigate the properties of blood cells, proteins, and other cell-scale entities that may be present in blood samples or other tissue samples. Flow cytometers are machines that reduce blood samples (or other fluid samples) to single-cell streams and then probe the samples with laser beams. Depending on the size and shape of the cell (or other particle of interest) some light will be diffracted around the cell (this is referred to as "forward scatter"), while other light will reflect off the cell (yielding "side scatter"). The combination of forward and side scatter provides information about any one particular cell; in general, larger cells produce greater forward scatter, and more complex cells (complexity in this context is often referred to as "granularity," measuring the geometric intricacy of the surfaces within or around cells which would scatter light waves in different directions) produce greater side scatter.

Note that these two dimensions are independent in general; cells can be both relatively large and relatively complex, so that greater forward scatter does not imply lesser side scatter, or vice-versa. In addition to laser probes, modern-day flow cytometry machines often use a diverse range of fluorochromes to mark cells or materials that may be present inside or on the surface of cells, which then emit fluorescent light signals that can be absorbed by detectors which are placed parallel to the photodiodes and photomultipliers that measure forward/side scatter. Collectively, the scatter signals and fluorochromes are called "channels." The data produced by modern flow cytometry instruments may have as many as 12 to 14 channels. Computationally, the end result is a data structure that is notated as a matrix, where each row corresponds to a single cell (referred to in coding as an "event") and each column represents one channel.

The canonical format for representing flow cytometry data is FCS (Flow Cytometry Standard). Each FCS file encodes the "event matrix" (essentially a table marking the observations for each cell according to each channel; this data has no relation to matrices in the linear-algebraic sense, except that code libraries designed for matrix arithmetic are convenient tools for representing the relatively large tables that emerge from FCS data).

Ultimately, the purpose of FCS representation is to cluster events (that is, individual cells) into groups and then count the relative size of each group, which can give information about the sample being analyzed; for instance, a blood sample can be examined to measure the ratio of monocytes to lymphocytes. Such classification does not emerge automatically from the raw event data; instead, a series of mathematical and geometrical transformations is often necessary to transform event matrices into scatterplots where event classification becomes visually obvious. The process of clustering events subsequent to these transformations is called "gating." As a result, a complete description of FCS analysis requires the raw FCS data to be paired with representations of transforms that are applied to individual channels and with descriptions of gates applied to the resulting two-dimensional spaces (each space being a side-by-side comparison of two channels). Flow cytometry data is also presented sometimes as a histogram, visualizing individual transformed channels on their own.

Note that this discussion has largely equated cells with events, given that, historically and in practice, the most common application of cytometry is to quantify the sorts of cells observed in a biological sample; however, cytometry can also be used to probe other submicroscopic (roughly cellular-scale) entities, such as microbes (which may be multi-cellular) or extracellular vessicles (particles that are released from cells). Flow cytometry can also indirectly examine entities that are too small to generate discrete events in the raw data; for instance, side-scatter and/or fluorescent channels can be used to quantify proteins or biomarkers occurring on the exterior of cells.

Preliminary to raw event data, FCS files also contain "headers" which present information about the individual channels and about the experimental setup (e.g., the make and model of the machine used for the actual cytometry). Headers in FCS are preceded by dollar-sign characters ($, or Latin-1 code 36). Some headers provide information about the overall experiment, whereas others are specific to individual channels. These latter header fields have a specific internal structure encoded in the tag name: in general they contain the letter **P** (for "parameter"), followed by a number representing the channel, and then an additional letter representing which specific piece of information is being presented about that channel. For example, the code **$P9F** would be used as a tag marking a string value representing the name of the optical filter applied to the 9th channel. The FCS standard also allows for post-processing information, such as channel transformations and gating, to be recorded in FCS headers fields, although such information is often instead presented in separate files, with formats such as GATINGML.

There are a variety of mathematical transforms that may be applied to channels prior to gating, including various logarithmic transforms, and bi-exponential calculations (such as transforms based on hyperbolic sines and inverse hyperbolic signs). Considering an overall flow cytometry workflow, there are in totality, then, at least four different stages of observation/analysis which must be recorded: machine setup and overall experimental properties (encoded via FCS header data); event matrices themselves; transform functions that have been applied to various channels; and gating steps, or the geometry of regions applied to cross-channel scatter plots that classify events based on cell types.

These different kinds of data have generally given rise to several different file formats, as well as to proposals to unify these formats into an overarching common vocabulary, potentially replacing FCS (as we will cite specifically in later chapters). One proposal is to encode FCS within DICOM; other suggestions are oriented around XML. The rationale for merging FCS within the overall DICOM infrastructure derives in part from the conceptual similarities between "gating" applied to FCS scatter-plots and "segmentation" of biomedical image data—although FCS plots are not images *per se*, they are akin to bioimages in that they are obtained through optical instruments and can (with suitable mathematical transformation) take on a visual form where clusters of related events (evincing a common cell-type) are geometrically circumscribed, roughly analogous to how discrete objects form distinguished regions within photographs. These conceptual similarities have yielded calls to record FCS data according to annotations and terminologies associated with image data, which is the subject of the next subsection.

2.3.4 Image segmentation, contours, and regions of interest

Many image-processing use-cases in biology and medicine involve isolating particular "foreground" regions from within diagnostic or laboratory images (e.g., individual cells from within a tissue sample magnified via microscopy; tumors within a radiographic image intended to diagnose solid-tumor cancer; or patterns of ground-glass opacity which suggest the presence of SARS-CoV-2 in lung scans). The process of isolating particular regions within images is sometimes discussed via the generic term "segmentation," although technically segmentation involves a complete partition of the image, wherein every point of the image is assigned to exactly one segment. The goal of image-analysis is often less about segmenting extraneous "background" material, but rather about demarcating *Regions of Interest,* which are conceptually more important than the rest of the image.

This overall theme applies outside of biology as well; image tagging refers to the process of classifying images according to their most prominent features or objects. If a photograph is classified as an image of a car, for example, the most important "Region of Interest" is the area which demarcates the car itself, considered as a foreground, from other background content (perhaps the street or driveway where the car is parked). Similarly, if a microscopic image reveals three blood cells, the regions of interest are the cells themselves.

In contrast to other discussions in this section, our examination of image-annotation will not focus on specific file formats so much as on how regions of interest may be characterized. To prepare this discussion, it is useful to consider a concrete example of how regions of interest may be isolated from images using image-processing algorithms. Readers may wish to consult the code accompanying book, which contains a demonstration of image analysis using the OpenCV Computer Vision library. This code sample demonstrates a common image-analysis workflow. The most crucial step is probably that of finding contours (via the OpenCV **findContours** procedure), which yield a set of pixel-sequences that (with some likelihood) mark boundaries between disparate objects in the source image. Usually contour-extraction is preceded by a preliminary step involving a "morphological operator," which has the effect of simplifying the image somewhat and reducing the chance that spurious boundaries will be discerned that are in fact effects of light or of inconsistent coloration.

After images have been transformed morphologically, boundaries are identified by searching for consistent color-patches on either side of an apparent boundary-line. The actual methodology for isolating apparent boundaries depends on the image-processing algorithm used. Most

algorithms work with matrix-based "kernels," which calculate color divergence one step away from an individual pixel. A given pixel, for instance, will in general be surrounded by eight other pixels (including those which are diagonally as well as orthogonally adjacent). In each of these directions, the adjacent pixel will diverge from the current pixel (at the center of the matrix) by some vector of quantities, depending on the color model. For example, in a conventional RGB (or BGR) color-space, two pixels may be compared in the red, green, and blue channels, so the contrast between adjacent pixels may be split into a vector of three magnitudes.

Such comparisons can then be repeated over each of the eight pixels surrounding a central point, yielding a 3-by-3 matrix each of whose entries contains three quantities (for red, green, and blue offsets). These matrices are typically reworked by applying some mathematical function to the matrix cells, often resulting in a single real-valued matrix (instead of a matrix of color vectors). The preferred calculation is then repeated at every point in the image, yielding a space of matrices which can be statistically analyzed; the end result of that analysis, at least in the context of boundary-detection algorithms, is a tracing of lines which appear to demarcate borders between distinct regions of an image, where "region" in this context refers to pixels grouped together by virtue of emanating from a common real-world object depicted via the image. Within image-analysis tools, such as OpenCV, such boundary-lines are referred to as "contours."

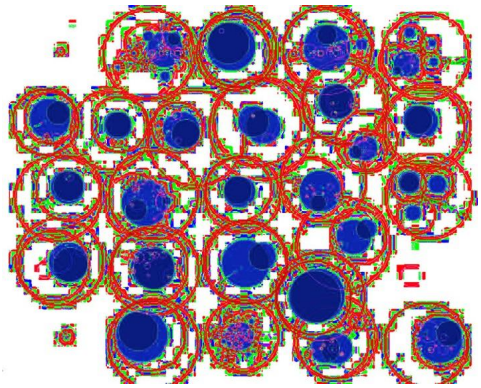
Once contours are extracted, OpenCV has a variety of tools that may be used to characterize the regions thereby delineated. These include the dimensions of the bounding rectangle (the smallest rectangle with sides parallel to those of the image overall that fully contains the contour) as well as dimensions and angles of the bounding rotated rectangle (the smallest rectangle containing the contours when the rectangle's sides are not restricted to being parallel with the external image boundaries).

Other metrics involve inscribed circles (the largest circle wholly contained within the contour) or circumscribing circles (the smallest circle wholly containing the contour) where either may be measured both at the center of gravity of the contour's internal region (that is, the circles are fixed at that point) or allowing the circle center to vary; one then looks for the largest inscribing circle, or smallest circumscribing circle, where the center point can be anywhere within or even (in the latter case) outside the contour's interior. Metrics based on circles may be generalized to ellipses, yielding further details based on the ellipses' eccentricity and the angle of their major axis.

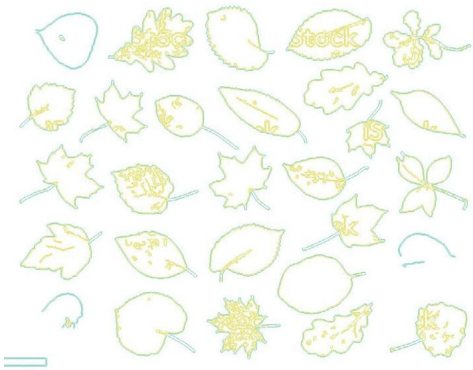
Separate and apart from these morphological analyses—which approximate the contour with canonical shapes, such as rectangles, circles, or ellipses—other methods quantify the degree of "convexivity" evinced by a contour (how much it deviates from its convex hull), the color-average inside the contour, the contour's area or perimeter, as well as relationships between contours (notably whether contours enclose other contours in their interior). The demo code shows calculations based on each of these measures. Via analyses such as these, every contour may be associated with space of at least 12–15 features, which in turn may be used to classify the regions bounded by each contour (a simple illustration of several contour-based metrics is rendered in Fig. 2.2). Not every contour is in fact the boundary of a bonafide image region; a principal goal of image-analysis is to isolate those contours which are likely to be statistically significant (in the sense of bounding a region of interest) as opposed to background "noise."

All of these metrics classify regions only on basic mathematical levels; none of this quantitative data suffices to discern whether contour-delimited regions correspond to cells, cars, or any other kind of real-world objects. Nevertheless, region feature-vectors are the starting

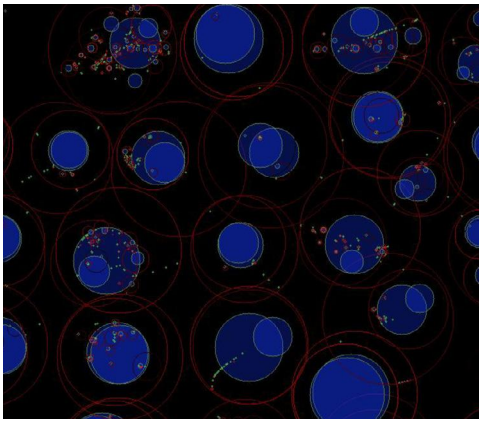
(a) original



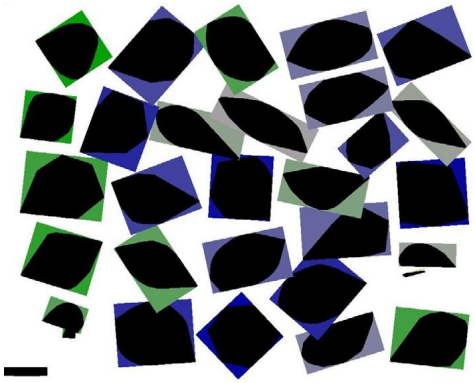
(b) inscribed circles (color inverted)



(c) filtered contours (color inverted)



(d) insribed circles (enlarged)



(e) convex hull with bounding rectangles

(f) interior-contour color average

FIGURE 2.2 Visualizing contour feature-vectors (note that in (e) the colors are inverted, and the gap between the convex hull and bounding rectangle is shaded to visualize the angle of the rectangle to external image borders).

point for statistical calculations that lead to more sophisticated image-tagging. This is roughly a two-step process, first identifying regions which seem likely to correspond to real-world objects and then classifying those objects themselves. In the biomedical context, image-tagging involves classifying regions that are biologically and/or diagnostically significant. An image-processing algorithm might attempt to isolate a solid-tumor mass, for example; such an algorithm may be able to correctly demarcate the region of an image where a tumor is visible. For diagnostic purposes further details would then need to be added, classifying tumors into categories such as "sarcomas larger than 5 cm." Such classifications may or may not be possible with automated tools; traditionally these details are added by radiologists or pathologists who manually examine radiographic images, although image-analysis tools could be used to expedite this process (e.g., to select the most diagnostically valuable images from a **DICOM** image-series).

The end result of biomedical image-analysis is of course these diagnostic or investigative annotations, such as labeling an image-region as a particular kind of tumor, with a particular size and characteristics—these are the details generally recorded in an annotated imaging database or notated via **DICOM** extensions, such as treatment plans or structured reports. Data of a more quantitative sort, such as the feature-vectors characterizing image contours, are important only as an intermediary processing step. Nevertheless, there are several reasons why intermediate image-analysis data may continue to be important even after the higher-level and more "semantic" (as opposed to purely quantitative) analytic steps have occurred.

For one thing, it is inconvenient for pathologists/radiologists to manually mark off image regions in the course of reporting their diagnostic findings. As such, machine-segmented Regions of Interest (or some subspace derived from them, such as a polygonal frame enclosing their convex hull) can be integrated into diagnostic reports, which means that techniques must be applied to incorporate mathematically extracted contours/regions into image annotations. Moreover, there are scenarios where behind-the-scenes image-processing data may need to be retained for subsequent research; this data may be relevant for holistic statistical analysis of large-scale data sets and/or for revisiting diagnoses over time. For instance, a software component may try to measure the progression of a tumor by comparing automatically extracted image regions taken at two different times, assuming that the corresponding regions depicting "the same" tumor can be isolated, and then mathematically compared.

For these kinds of scenarios, it is reasonable to assume that at least some intermediate image-processing data should be retained for subsequent analysis, which then raises the question of how image-feature vectors should optimally be represented in a database context. We will address this question in a later chapter.

Similar comments apply also to audio feature-vectors, such as those derived from fast Fourier transforms (**FFT**) or mel-frequency cepstral coefficients (**MFCC**), which in bioinformatics may classify **EKG**s, sonograms, and other biologically useful audio resources; [18] asserted that for EKG signals "[m]orphological features include the coefficients of the Hermite transform, the wavelet transform or the discrete cosine transform ... that aim to model the ECG beat instead of extracting features from the raw data" (page 2). Because machine-learning and other statistical analytic techniques operate on **EKG** feature vectors, these vectors serve as intermediate representations of **EKG** waveforms from which diagnostic findings are derived. The process of extracting feature-vectors from audio waveforms bears some mathematical similarities to that of extracting feature-vectors from distinguished regions of bioimages (reflected in the extension of **DICOM** to encompass audio

assets), which will be emphasized in later chapters.

2.3.5 Common data models for clinical research

Other biomedically important data structures derive from initiatives to share or pool clinical data for research purposes. A good example of such an initiative is the OMOP partnership mentioned earlier. The OMOP format prioritizes data integration according to largely conventional SQL paradigms; in the OMOP context, most information is held in relational databases, and data integration involves merging disparate database tables. Toward that end, the common data model standardizes those specific tables that should be present among compliant data sets and the columns within each of these tables, where the various columns are given a canonical name and data type. For example, specific tables include ones for personal (patient) information, data about a specific visit made by a patient to a health-care provider, data about one specific procedure administered to a patient, information about a specific occasion when a patient received a drug/pharmaceutical, data about the costs of either drugs or procedures (or other "medical events"), and so forth. The point is not that health care providers' software would internally store information with this precise configuration of tables, but rather that biomedical data (in whatever format internally used) can be shared by distributing each data field into the specific collection of tables and columns mandated by the OMOP CDM. The key data-integration step in this case is then whichever algorithms each individual institution uses to initialize OMOP tables from local information.

Readers who would like to examine the specific tables and columns which constitute the OMOP Common Data Model may consult the source code accompanying this book (see the **omop** project). This code implements C++ classes corresponding to OMOP CDM tables. The project is not an OMOP client as such—there is no logic for populating the tables from a database instance—but it serves as a reference illustrating the fields and structures that are endemic to OMOP data representations.

The accompanying code similarly has a **pcor** project, which demonstrates the PCORNET Common Data Model. Similar to the OMOP CDM, this standard (developed by the National Patient-Centered Clinical Research Initiative) promotes data integration by specifying that shared data be distributed into a collection of tables with fixed roles and column-sets. The various tables include a "Demographic" specification with essential personal patient info, a table representing patients' enrollment in a clinical trial, a "Dispensing" table noting medicines dispensed by a pharmacy to a patient, and tables for diagnoses, lab results, "Vital Signs," and so forth. It is interesting to contrast the OMOP and PCORNET Common Data Models, which can be readily compared because they are structurally similar; the fact that they nonetheless recognize fairly different table and column sets point to the specific interests of observational medicine and of patient-centered advocacy, respectively.

In contrast to the table-based and SQL-inspired formats for OMOP and PCORNET, parts of the CDISC standard lean more toward Semantic Web information structures (as with the BRIDG (Biomedical Research Integrated Domain Group) specification, which is one of several CDISC standard models). The contrast between table-oriented and Semantic Web notions of a "Controlled Vocabulary" embody alternative theories of the role of semantic normalization in technology. For table-oriented environments, controlled vocabularies specifically address the organization of digital resources; for example, table names, column names, and in some cases the values asserted within columns are required to have identifiers drawn from a restricted class of terms. This restriction applies to individual values when those values are neither numeric nor string types (such as a person's

name, which for obvious reasons cannot be tied to a list of accepted values) but rather to columns whose data may be reasonably restricted to a given enumeration. For instance, the state where a patient resides must be one of the fifty US states (assuming a patient is an American citizen and excluding, for sake of argument, territories such as the District of Columbia). It is possible to write or abbreviate state names in different ways, but a Controlled Vocabulary could reasonably stipulate that all conformant data sets use identical terms for the same state name.

This discussion uses terminology associated with relational database tables; for general discussion of data integration it makes sense however to adopt more type-theoretic language, so that we can consider the names of data types, the names of the fields that data types contain, and (in some cases) the values that are spanned by those fields. In lieu of restricting names to data types themselves, a system may instead assign names to *interfaces*, or, in effect, to prototypes and specifications which data types need to put into operation. A protocol may specify a collection of interfaces, and a software component which implements the protocol will provide data for each interface that the protocol outlines. Controlled Vocabularies therefore specify the names of interfaces, as well as the names of data fields accessed through the values that instantiate data types implementing each interface. Controlled Vocabularies would also specify the range of nominal values assigned to enumerative types, in the case of data types whose values derive from a specified list (e.g., the set of US states).

In the case of Semantic Web data, by contrast, Controlled Vocabularies (typically called "Ontologies" in this context) serve a more generic role, typically connecting data types not only to particular software implementations, but also to more abstract concepts. For example, cancer is classified (according to histological criteria) into one of six groups (Carcinoma, Sarcoma, Myeloma, Leukemia, Lymphoma and Mixed). A data-representation standard could reasonably stipulate that any tumor—that is, any data structure describing the characteristics of a tumor—include a data field classifying the tumor into one of the six groups (at least those whose cancers present as tumors that can be detected via bioimaging). In this sense the six groups (and canonical names associated with them) constrain the range of values spanned by a cancer-type (and by extension tumor-type) data field. Such a formal stipulation can be defined both as a restriction on table data and as part of a Semantic Web Ontology; in short, semantic standardization at this level is common to both paradigms. However, an ontology may further model "philosophical" traits associated with a tumor; for instance, the fact that a tumor is a form of biological tissue, and more generally a biological entity, which in turn is a physical thing, possessing characteristics such as size and volume. These more general concepts do not fit within type systems themselves—there is rarely a data type modeling "physical things" in general—so the purpose of standardized vocabularies in contexts, such as the Semantic Web is not merely to serve as a guideline for implementing code libraries (e.g., those that would export data in the **OMOP** or **PCORNET** Common Data Models), but as a basis for artificial intelligence. In this sense the values that are represented within Semantic Web data structures may in some cases be associated with specific computational data types, but they are also modeled in terms of networks of concepts that are designed to serve as a foundation for **AI** reasoning.

Later in the book, we will return to the role of Ontologies and other meta-models in data-integration contexts. Before engaging these more theoretical discussions, however, we will try to develop a more substantial concrete picture of how data-integration problems arise. This discussion will be the focus of the following chapter, beginning with further consideration of

the overlap between data-integration challenges and precision medicine.

References

[1] Fedaa Ali, et al., The new SARS-CoV-2 strain shows a stronger binding affinity to ACE2 due to N501Y mutation, Briefings in Bioinformatics 22 (2021), https://arxiv.org/pdf/2101.01791.pdf.

[2] Ali F. Alsulami, et al., SARS-CoV-2 3D database: understanding the coronavirus proteome and evaluating possible drug targets, Briefings in Bioinformatics 22 (2) (2021) 769–780, https://academic.oup.com/bib/article-pdf/22/2/769/36655605/bbaa404.pdf.

[3] Donald J. Benton, et al., The effect of the D614G substitution on the structure of the spike glycoprotein of SARS-CoV-2, Proceedings of the National Academy of Sciences of the United States of America 118 (9) (2021), https://www.pnas.org/content/118/9/e2022586118.

[4] J.C. Blader, et al., Can keeping clinical trial participants blind to their study treatment adversely affect subsequent care?, Contemporary Clinical Trials 26 (3) (2005) 290–299, https://europepmc.org/article/med/15911463.

[5] Yongfei Cai, et al., Distinct conformational states of SARS-CoV-2 spike protein, Science 369 (2020) 1586–1592, https://science.sciencemag.org/content/369/6511/1586.

[6] Che-Mai Chang, et al., Profiling of T cell repertoire in SARS-CoV-2-infected COVID-19 patients between mild disease and pneumonia, Journal of Clinical Immunology 41 (2021) 1131–1145, https://link.springer.com/article/10.1007/s10875-021-01045-z.

[7] Sandipan Chakraborty, Evolutionary and structural analysis elucidates mutations on SARS-CoV-2 spike protein with altered human ACE2 binding affinity, Biochemical and Biophysical Research Communications 534 (2021) 374–380, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7699163/pdf/main.pdf.

[8] Bethany Dearlove, et al., A SARS-CoV-2 vaccine candidate would likely match all currently circulating variants, View ORCID Profile, Proceedings of the National Academy of Sciences of the United States of America 117 (2020) 23652–23662, https://www.pnas.org/content/117/38/23652.

[9] Liangwei Duan, et al., The SARS-CoV-2 spike glycoprotein biosynthesis, structure, function, and antigenicity: implications for the design of spike-based vaccine immunogens, Frontiers in Immunology (2020), https://www.frontiersin.org/articles/10.3389/fimmu.2020.576622/full.

[10] Katja Fink, Can we improve vaccine efficacy by targeting T and B cell repertoire convergence?, Frontiers in Immunology (2019), https://www.frontiersin.org/articles/10.3389/fimmu.2019.00110/full.

[11] Natalia T. Freund, et al., Multi-clonal SARS-CoV-2 neutralization by antibodies isolated from severe COVID-19 convalescent donors, PLoS Pathogens (2021), https://journals.plos.org/plospathogens/article?id=10.1371/journal.ppat.1009165.

[12] Simon Friedensohn, et al., Synthetic standards combined with error and bias correction improve the accuracy and quantitative resolution of antibody repertoire sequencing in human naive and memory B cells, Frontiers in Immunology (2018) (special issue on "B Cell Biology"), https://www.frontiersin.org/Journal/10.3389/fimmu.2018.01401/full.

[13] Jacob D. Galson, et al., Deep sequencing of B cell receptor repertoires from COVID-19 patients reveals strong convergent immune signatures, Frontiers in Immunology (2020), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7769841.

[14] Rishi R. Goel, et al., Distinct antibody and memory B cell responses in SARS-CoV-2 naive and recovered individuals after mRNA vaccination, Science Immunology 6 (58) (2021), https://immunology.sciencemag.org/content/6/58/eabi6950.

[15] Cheolmin Kim, et al., A therapeutic neutralizing antibody targeting receptor binding domain of SARS-CoV-2 spike protein, Nature Communications 12 (2021), https://pubmed.ncbi.nlm.nih.gov/33436577.

[16] Bette Korber, et al., Tracking changes in SARS-CoV-2 spike: evidence that D614G increases infectivity of the COVID-19 virus, Cell 182 (2020) 812–827, https://www.cell.com/cell/pdf/S0092-8674(20)30820-5.pdf.

[17] Laura López-Santibáñez-Jácome, et al., The pipeline repertoire for Ig-seq analysis, Frontiers in Immunology (2019), https://www.frontiersin.org/articles/10.3389/fimmu.2019.00899/full.

[18] Aurore Lyon, et al., Computational techniques for ECG analysis and interpretation in light of their contribution to medical advances, Journal of the Royal Society Interface 15 (2018), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5805987.

[19] Claire Marks, Charlotte M. Deane, How repertoire data are changing antibody science, Journal of Biological Chemistry 295 (2020) 9823–9837, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7380193.

[20] Ivan Mercurio, et al., Protein structure analysis of the interactions between SARS-CoV-2 spike protein and the human ACE2 receptor: from conformational changes to novel neutralizing antibodies, Cellular and Molecular Life Sciences 78 (4) (2021) 1501–1522, https://pubmed.ncbi.nlm.nih.gov/32623480.

[21] Anastasia A. Minervina, et al., Longitudinal high-throughput TCR repertoire profiling reveals the dynamics of T-cell memory formation after mild COVID-

19 infection, eLife (2021), https://elifesciences.org/articles/63502.
[22] Anshumali Mittal, et al., COVID-19 Pandemic: insights into structure, function, and hACE2 receptor recognition by SARS-CoV-2, PLoS Pathogens (2020), https://journals.plos.org/plospathogens/article?id=10.1371/journal.ppat.1008762.
[23] Anwar Mohammad, et al., Higher binding affinity of furin for SARS-CoV-2 spike (S) protein D614G mutant could be associated with higher SARS-CoV-2 infectivity, International Journal of Infectious Diseases 103 (2021) 611–616, https://www.ijidonline.com/article/S1201-9712(20)32237-2/pdf.
[24] Hanh T. Nguyen, et al., Spike glycoprotein and host cell determinants of SARS-CoV-2 entry and cytopathic effects, Journal of Virology 95 (5) (2021), https://journals.asm.org/doi/pdf/10.1128/JVI.02304-20.
[25] Xiuyuan Ou, et al., Characterization of spike glycoprotein of SARS-CoV-2 on virus entry and its immune cross-reactivity with SARS-CoV, Nature Communications 11 (2020), https://www.nature.com/articles/s41467-020-15562-9.
[26] Victor Padilla-Sanchez, *In silico* analysis of SARS-CoV-2 spike glycoprotein and insights into antibody binding, Research Ideas and Outcomes 6 (2020), https://riojournal.com/article/55281/.
[27] Utsav Pandey, et al., High prevalence of SARS-CoV-2 genetic variation and D614G mutation in pediatric patients with COVID-19, Open Forum Infectious Diseases 8 (6) (2020), https://academic.oup.com/brain/advance-article/doi/10.1093/brain/awaa240/5868408.
[28] Suman Pokhrel, et al., Increased elastase sensitivity and decreased intramolecular interactions in the more transmissible 501Y. V1 and 501Y. V2 SARS-CoV2 variants' spike protein – an *in silico* analysis, PLoS ONE 16 (2021), https://doi.org/10.1371/journal.pone.0251426.
[29] Supriya Ravichandran, et al., Longitudinal antibody repertoire in "mild" versus "severe" COVID-19 patients reveals immune markers associated with disease severity and resolution, Science Advances 7 (19) (2021), https://advances.sciencemag.org/content/7/10/eabf2467.
[30] Christoph Schultheiß, et al., Next-generation sequencing of T and B cell receptor repertoires from COVID-19 patients showed signatures associated with severity of disease, Immunity 53 (2) (2020) 442–445, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7324317.
[31] Osman Shabir, The phylogenetic tree of the SARS-CoV-2 virus, News Medical, https://www.news-medical.net/health/The-Phylogenetic-Tree-of-the-SARS-CoV-2-Virus.aspx.
[32] Gentle Sunder Shrestha, et al., Precision medicine for COVID-19: a call for better clinical trials, Critical Care 24 (2020), https://ccforum.biomedcentral.com/articles/10.1186/s13054-020-03002-5.
[33] Giwan Seo, et al., Rapid detection of COVID-19 causative virus (SARS-CoV-2) in human nasopharyngeal swab specimens using field-effect transistor-based biosensor, ACS Nano 14 (4) (2020) 5135–5142, https://pubs.acs.org/doi/10.1021/acsnano.0c02823.
[34] Joshua B. Singer, et al., GLUE: a flexible software system for virus sequence data, bioRxiv, https://www.biorxiv.org/content/10.1101/269274v3.full.pdf, 2018.
[35] Liz Szabo, Unraveling the mysterious mutations that make delta the most transmissible COVID virus yet, Quartz (2021), https://qz.com/2039772/why-delta-variant-spreads-twice-as-fast.
[36] Matthieu Van Tilbeurgh, et al., Predictive markers of immunogenicity and efficacy for human vaccines, Vaccines 9 (6) (2021), https://pubmed.ncbi.nlm.nih.gov/34205932.
[37] Martijn M. VanDuijn, et al., Immune repertoire after immunization as seen by next-generation sequencing and proteomics, Frontiers in Immunology (2017), https://www.frontiersin.org/articles/10.3389/fimmu.2017.01286/full.
[38] Gennady M. Verkhivker, et al., Landscape-based mutational sensitivity cartography and network community analysis of the SARS-CoV-2 spike protein structures: quantifying functional effects of the circulating D614G variant, ACS Omega 6 (24) (2021) 16216–16233, https://pubmed.ncbi.nlm.nih.gov/34179666.
[39] Erik Volz, et al., Evaluating the effects of SARS-CoV-2 spike mutation D614G on transmissibility and pathogenicity, Cell 184 (2021) 64–75, https://www.sciencedirect.com/science/article/pii/S0092867420315373.
[40] William N. Voss, et al., Prevalent, protective, and convergent IgG recognition of SARS-CoV-2 non-RBD spike epitopes, Science 372 (2021) 1108–1112, https://science.sciencemag.org/content/372/6546/1108.
[41] Aleksandra M. Walczak, et al., Inferring the immune response from repertoire sequencing, PLOS Computational Biology (2020), https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007873.
[42] Alexandra C. Walls, et al., Structure, function, and antigenicity of the SARS-CoV-2 spike glycoprotein, Cell 184 (2021) 64–75, https://www.sciencedirect.com/science/article/pii/S0092867420302622.
[43] Yasunori Watanabe, et al., Native-like SARS-CoV-2 spike glycoprotein expressed by ChAdOx1 nCoV-19/AZD1222 vaccine, ACS Central Science 7 (4) (2021) 594–602, https://pubs.acs.org/doi/10.1021/acscentsci.1c00080.
[44] Anna Winger, Thomas Caspari, The spike of concern – the novel variants of SARS-CoV-2, Viruses 13 (6) (2021), https://www.mdpi.com/1999-4915/13/6/1002/pdf.

[45] Ian D. Wolfe, et al., Exploration and ethical analysis of open-label pediatric vaccine trials in a pandemic, Clinical Therapeutics (2021), https://www.sciencedirect.com/science/article/abs/pii/S0149291821002046.

[46] Yanling Wu, et al., Deep mining of human antibody repertoires: concepts, methodologies, and applications, Small Methods (2020), https://onlinelibrary.wiley.com/doi/full/10.1002/smtd.202000451.

[47] Qiongqiong Angela Zhou, et al., Potential therapeutic agents and associated bioassay data for COVID-19 and related human coronavirus infections, ACS Pharmacology & Translational Science 3 (5) (2020) 813–834, https://pubs.acs.org/doi/10.1021/acsptsci.0c00074.

[48] Lizhou Zhang, et al., SARS-CoV-2 spike-protein D614G mutation increases virion spike density and infectivity, Nature Communications 11 (1) (2020), https://pubmed.ncbi.nlm.nih.gov/33243994.

[49] Shi Zhao, et al., Modelling the association between COVID-19 transmissibility and D614G substitution in SARS-CoV-2 spike protein: using the surveillance data in California as an example, Theoretical Biology and Medical Modelling (2020), https://tbiomed.biomedcentral.com/articles/10.1186/s12976-021-00140-3.

[50] Guanghong Zuo, et al., CVTree: a parallel alignment-free phylogeny and taxonomy tool based on composition vectors of genomes, bioRxiv, https://www.biorxiv.org/content/10.1101/2021.02.04.429726v1.full.pdf, 2021.

This page intentionally left blank

CHAPTER

3

Data mining and predictive analytics for cancer and COVID-19

OUTLINE

3.1 Introduction

This chapter will touch on research methodology and clinical trial design in light of patient-centered paradigms and precision medicine. We will focus in particular on bioimaging/radiology, on trial architecture (particularly vis-à-vis COVID-19), and on text and data mining.

After examining how precision medicine affects requirements for bioimaging software, we will consider patient-centered priorities in the context of clinical trials. The goal of incorporating more granular patient-specific data within clinical observations in general translates over to clinical trials in particular, with trial-specific data models needing to incorporate patient-

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00008-8

centered details at several stages, including recruitment of patients for trials, information management while trials are being conducted, and subsequent follow-up studies and/or analyses.

This chapter will conclude by examining text and data mining techniques oriented toward biomedical publications and data sets, on the premise that text and data mining represent a computationally sophisticated methodology in their own right. We will focus, as a case study, on **CORD-19**, a large corpus of open-access COVID-related articles.

3.2 Precision medicine and bioimaging

Patient-centered research in the radiological context is centered on improving the precision of diagnostic-imaging techniques and corresponding clinical interventions. Indeed, the goal of contemporary radiology is not only to confirm diagnoses, but also to extract cues from medical images that suggest which course of treatment has the highest probability of favorable outcomes. A related goal is curating collections of diagnostic images so as to improve our ability to identify such diagnostic and prognostic cues, potentially using machine learning and/or artificial intelligence applied to large-scale image repositories.

The goal of building "searchable" image repositories has inspired projects such as the Semantic DICOM Ontology (**SeDI**)[1] and the **VISION** "structured reporting" system.[2] As explained in the context of **SeDI**:

> [If] a user has a CT scan, and wants to retrieve the [corresponding] radiation treatment plan ... he has to search for the RTSTRUCT object based on the specific CT scan, and from there search for the RTPLAN object based on the RTSTRUCT object. This is an inefficient operation because all RTSTRUCT [and] RTPLAN files for the patient need to be processed to find the correct treatment plan. *[37, page 1]*

Even relatively simple queries, such as "display all patients with a bronchial carcinoma bigger than 50 cm^3", cannot be processed by **PACS** systems: "although there are various powerful clinical applications to process image data and image data series to create significant clinical analyses, none of these analytic results can be merged with the clinical data of a single patient."[3] These limitations partly reflect the logistics of how information is transferred between clinical institutions and radiology labs. In response, and in an effort to advance the science of diagnostic image-analysis, organizations such as the Radiological Society of North America (**RSNA**) have curated open-access data sets encompassing medical images as well as image-annotations (encoding feature vectors) that can serve as reference sets and test corpora for investigating analytic methods. Such repositories are designed to integrate data from multiple hospitals and multiple laboratories—bypassing the conventional data flows, wherein radiological information is shared between clinicians and radiologists, but is not also merged into broad-spectrum corpora.

This renewed focus on patient outcomes has intriguing consequences for the scope and requirements of diagnostic-imaging software. In particular, the domain of radiological applications is no longer limited to **PACS** workstations where pathologists perform their diagnostic analysis, with the results transferred back to the referring institution (and subsequently available only through that institution's medical records, if at all). In the older, conventional workflow, radiographic images are requested by some medical institution for diagnostic purposes. Relevant information is therefore shared between two end-points: the institution that pre-

[1] See https://bioportal.bioontology.org/ontologies/SEDI.
[2] See https://epos.myesr.org/esr/viewing/ index.php?module=viewing_poster&task=&pi=155548.
[3] See https://semantic-dicom.com/starting-page/.

scribes a diagnostic evaluation and the radiologist or laboratory that analyzes the resulting images. Building radiographic data repositories complicates this workflow, because a third entity becomes involved; the organization responsible for aggregating images and analyses is generally distinct from both the prescribing institution and the radiologists themselves. As a result, both radiologists and prescribing institutions, upon participation in the formation of the target repository, must identify which image series and which patient data are proper candidates for the relevant repository.

For a concrete example, **RSNA** has announced the forthcoming publication of an open-access image repository devoted to COVID-19 (specifically the organization established two "task forces" in 2020; as of mid-2021 the overall project appears still to be under development, although several recent studies on COVID-19 in general have been published in **RSNA** journals).[4] This repository is being curated in collaboration with multiple European, Asian, and South American organizations so as to collect data from hospitals treating COVID-19 patients. Such a collaboration requires protocols both for data submission and for patient privacy and security.

3.2.1 The basic synthesis between bioimaging and precision medicine

Patient-centered research in the radiological context has several dimensions, including analysis of the rationales for diagnostic imaging in the first place: How well does image-based diagnostics actually correlate with improved patient outcomes? How can we quantify the experience of image-based testing itself (e.g., to identify factors such as cost, discomfort, and radiation danger which can rank some tests as less desirable than others, as one parameter to consider when deciding whether to prescribe imaging, and which modality)? These were among the questions addressed by the Patient-Centered Outcomes Research Institute's (**PCORI**'s) "Patient-Centered Research for Standards of Outcomes in Diagnostic Tests" (**PROD**) study [43], which also established a useful protocol for how medical institutions could provide reports on the experience and effectiveness of imaging from a patient's as well as clinician's perspective.

Assuming diagnostic imaging of a given modality *is* appropriate, an additional priority should then be to structure the diagnostic workflow—the image procurement, analysis, and data/metadata sharing protocols—to maximize the probability that subsequent clinical interventions are chosen which promote favorable outcomes on multiple patient-centered criteria, including quality of life and patient engagement. In the best case scenario, the goal of radiology is not only to confirm a diagnosis, but also to extract cues from medical images that suggest which course of treatment has the highest probability of favorable treatment outcomes.

Although most radiologists and pathologists would probably agree with this assessment—that in the best-case scenario image-processing can yield predictive analytics that would sort patient populations into groups steered toward distinct treatments deemed most likely to be effective—there are technological and operational challenges to making patient-centered perspectives a central feature of diagnostic-imaging methodology. Effectively cross-referencing imaging and outcomes data requires integrating heterogeneous information obtained at different times and places. Some clinical data is associated with each patient at the time that a radiological (or other imaging) study is prescribed. The images themselves, and subsequent diagnostic reports, then provide layers of data that exist prior to the initiation of a course of treatment. Moreover, a rigorous data-integration protocol would need to incorporate information emerging *after* the treatment starts: descriptions and assessments of clinical outcomes, and, po-

[4] See https://www.rsna.org/covid-19.

tentially, new data garnered by applying different image-analysis techniques. In brief, data-management protocols must be in effect before, during, and after image-analysis itself.

Ideally, image analysis can be powerful enough not only to identify a pathology, but to classify diagnoses into clusters based on recommended course of treatment. For analytic techniques to achieve this level of detail, however, it is necessary to arrive at a feedback loop where known clinical outcomes are associated with prior images so that developers have those outcomes available as a further dimension of clinical data that may be statistically cross-referenced with image analysis.

The PROD study demonstrates that experiential factors should be evaluated—both in terms of testing itself (and its risks/costs) and in terms of post-diagnostic quality of life—as part of the data modeling treatment outcomes, and the comparative effectiveness of a selected course of treatments compared to alternative diagnostic methods and/or clinical interventions. From the perspective of standard data models, initiatives to cross-reference imaging and outcomes data include several Semantic Web Ontologies, such as the Semantic DICOM Ontology (SeDI) and the ViSion "structured reporting" system (both referenced earlier). The purpose of these ontologies is to standardize the terms through which radiographic procedures, analyses, and recommendations are described more precisely or predictably than such older technologies as DICOM headers, DICOM-RT, and the RadLex lexicon. By properly aligning image metadata spanning multiple patients, it is possible to create "searchable" image archives such that images can be selected or classified within a larger image collection, yielding image series or patient cohorts that can be studied through the lens of predictive modeling or patient-centered outcomes. Projects such as SeDI implement "semantic" PACS workstations, where the space of known images is defined by a particular PACS system, but analogous techniques could be used to construct larger-scale image corpora as well, for research purposes, data mining, or as test-beds for code and algorithms. Patient-centered data points, such as those formulated via PROD, may then be incorporated as supplemental data.

3.2.2 Case study: the cancer imaging phenomics toolkit

At the forefront of diagnostic imaging are new analytic techniques that statistically analyze image data, often detecting signals that are invisible to the naked eye, or at least processing groups of image-series on a scale beyond the reach of human radiologists. The proliferation of image-analysis methodologies places a new emphasis on *extensibility*, in which the capabilities of bioimaging software can be expanded in a decentralized, modular fashion.

A representative example of this new paradigm is CaPTk (cancer imaging phenomics toolkit), which provides a primary component supplying a centralized user interface and taking responsibility for acquiring and loading radiographic/microscopy images, paired with multiple "peer" applications which can be launched from CaPTk's main window, or alternatively used as standalone programs [10, especially section 4]. In particular, CaPTk provides an implementation (apparently the only C++-based implementation) of the Common Workflow Language (CWL), using this workflow model in conjunction with the Qt reflective programming system to implement workflows connecting the central CaPTk application with its analytic extensions.[5] In effect, CaPTk achieves a workflow and messaging protocol for what they term "na-

[5]No native C or C++ libraries are described on the CWL website among the tools and parsers available for CWL, but CaPTk is mentioned on a corresponding discussion thread concerning C++ libraries. It seems therefore that the CaPTk "utilities" repository provides the de-facto standard C++ implementation of CWL, at least according to the CWL group themselves.

tive," "standalone" applications, yielding an extensible architecture through which new image-analysis techniques can be integrated into an underlying PACS system (a detailed discussion of CaPTk module implementation is outside the scope of this chapter, but this book's supplemental materials include a more technical overview of CaPTk; for now we just use CaPTk as a case study in modular design for bioimaging).

Taking the RSNA COVID-19 repository as a case study for promoting research into post-diagnostic outcomes, this repository is possible because an international team of hospitals and institutions have agreed to pool radiological data relevant to SARS-CoV-2 infection according to a common protocol. Taking CaPTk as a case study in multi-modal image analysis, this system is likewise possible because analytic modules can be wrapped into a plugin mechanism that allows many different algorithms to be bundled into a common software platform. Of course, these two areas of data-sharing overlap: one mission of image repositories, such as the RSNA's, is to permit many different analyses to be performed on the common image assets. The results of these analyses then become additional information, which enlarges the repository proportionately.

The CaPTk project serves as a useful case study for patterns in the analytic convergence or cross-referencing between image-analysis and outcomes/patient-centered data because it is extensible as part of its essential design (although in terms of large-scale adoption more conventional PACS clients may also be usefully examined simply because CaPTk has certain software-engineering innovations that make it an outlier from an implementational point of view). Analysis of the aforementioned RSNA COVID-19 images, for example, could be enacted via specialized CaPTk modules, which could in turn be provisioned with patient outcomes and Comparative Effectiveness Research (CER) capabilities. That is, modular design at the image-processing level can be leveraged to incorporate CER and predictive-analytic information sources (such as clinical records and immunological profiles) alongside image data proper (such as feature vectors calculated via computer vision algorithms).

A further level of integration between CaPTk and CER initiatives (again, staying with CaPTk as a representative example of bioimaging applications in general) can be achieved if one observes that clinical outcomes may be part of the analytic parameters used by CaPTk modules. As presently constituted, CaPTk analytic tools are focused on extracting quantitative (or quantifiable) features from image themselves, without considering additional patient-centered context. There is no technical limitation, however, which would prevent the CaPTk system from sharing more detailed clinical information with its modules, allowing these analytic components to cross-reference image features with clinical or patient information. This then raises general questions about sharing clinical data *as well as* information derived from bioimage analysis, bringing us to more general themes in lab/clinical data-sharing.

3.2.3 Multi-application networks in the context of scientific research data

Architecturally, the pattern of organization just described—semi-autonomous applications linked together (often by virtue of being common extensions to an overarching "core" software platform)—is analogous to the collection of software components that may share access to a data repository or a research-data corpus, include a corpus of medical/diagnostic images. The purpose of research data archives—particularly when they embrace contemporary open-access standards such as FAIR (findable, accessible, interoperable, reusable) [36] and the Research Object protocol[6]—is to promote reuse

[6] See http://www.researchobject.org/scopes/.

and reproduction of published data and findings, such that multiple subsequent research projects could be based on data originally published to accompany one book or article. As a result, it is expected that numerous projects may overlap in their use of a common underlying data set, which potentially means a diversity of software components implementing a diversity of analytic techniques, each offering a unique perspective on the underlying data.

Implementing a robust research-data software framework involves integrating multiple scientific applications, but also (ideally) extending these applications with features specifically of interest to those conducting or reviewing research using published data sets and/or described in academic literature: for instance, capabilities to download data sets from open-access scientific portals; to parse microcitation formats; and to interoperate with document viewers. This review of data-publishing technology is relevant to radiology and to patient-centered outcomes because it typifies the emerging ecosystem where scientific research and open-access data is being disseminated. The architecture employed by CaPTk is a useful example of how multiple autonomous, standalone, native applications can be federated into a decentralized but unified platform, logistically embodying the kinds of application networks appropriate for the technology supporting archives of research data (including diagnostic-imaging repositories).

Initiatives such as Research Objects and FAIR advocate for a technological infrastructure characterized by a diverse software ecosystem paired with open-access research data sets [3]. Although formats such as Research Objects have been standardized over the last decade, there has not been a comparable level of attention given to formalizing how multiple software applications should interoperate when manipulating overlapping data. The Common Workflow Language (CWL), which has been explicitly included in the Research Object model, documents one layer of inter-application messaging, including the encoding of parameters via command-line arguments (as mentioned earlier, CaPTk provides the most complete C++ implementation of CWL, using it to pass initial data between modules). Serializing larger-scale data structures is of course a generic task of canonical encoding formats, such as JSON, XML, RDF, and protocol buffers—not to mention text or binary resources serialized directly from runtime objects via, for instance, **QTextStream** and **QDataStream.** This means that some level of inter-application communications is enabled via CWL, and that essentially any computationally tractable data structure can be encoded via formats such as XML. These solutions, however, are sub-optimal: XML (as well as JSON and analogous formats) is limited because it takes additional development effort to compose the code that marshals data between runtime and serial formats. Similarly, although CWL can model information passed between applications, it provides only an indirect guide for programmers implementing each application's "operational semantics"—that is, the procedures which must be executed before and after the event wherein data is actually passed between endpoints.

In the context of CaPTk, for example, integrating peer modules with the CaPTk core application involves more than simply ensuring that these endpoints communicate via a standardized data-serialization format: the plugins must be *registered* with the core application, which affects the core in several areas, including the build/compile process and construction of the main GUI window. Modeling the interconnections between semi-autonomous modules, as CaPTk demonstrates, therefore requires more detail than simply modeling their shared data encodings; it is furthermore necessary to represent all procedural and user interface requirements in each component that may be affected by the others. Despite the standardization efforts that have been invested in Research Objects and the Common Workflow Language, we con-

tend that this fully detailed protocol for multi-application interop has not yet been rigorously formalized.

Rigorous models of application networks among semi-autonomous components acquire an extra level of complexity precisely because of this intermediate status: protocol definitions have to specify both the functional interdependence and the operational autonomy of different parts of the application network. Although one application does not need detailed knowledge of the other's internal procedure signatures (which would break encapsulation), the functional interdependence between applications can accordingly be modeled by defining protocols that must be satisfied by procedure-sets internal to each end-point—the relevant information from an integrative standpoint is not the actual procedures involved, but confirmation that the relevant procedure sets adhere to the relevant multi-procedural protocol.[7]

While we have initially approached multi-application networking from the bioimaging perspective, this topic is equally applicable to multi-site trial architecture, the theme of next section.

3.3 Precision medicine in trial design

Converting basic research to clinical practice directly benefiting patients—sometimes called *translational informatics*, a "research cycle, which involves the translation of knowledge and evidence [to] provision of evidence-based care in the clinical or public health settings" [26, page 2]—is sometimes represented as a two-stage process, which first involves translating research to clinical trials, and then formulating point-of-care practices on the basis of trial results ([26, page 2]). Data-sharing initiatives need to pay particular attention to the logistics of translational informatics in contexts where granular patient-specific information is important, such as immunoprofiling. Questions that should be addressed include (1) where data is to be hosted; (2) how participating institutions should submit data to a central repository; (3) how participating institutions and/or outside investigators should access previously deposited data; (4) how to ensure anonymization of patient-specific records; (5) how to ensure that different labs used by different hospitals are utilizing compatible protocols, so that results from multiple labs/hospitals can be seamlessly merged in a shared data commons; (6) how to ensure proper alignment between software employed at different institutions; and (7) how to incorporate data curated within the context of a multi-institutional data-sharing initiative into scientific papers documenting research findings. Each of these areas of concern are technically demanding because of the complex and heterogeneous nature of immunological profiles.

As a case study in clinical-trial software engineering, consider again the proposals in Shrestha *et al.* [35], which we reviewed in the previous chapter. As these authors recommend, COVID-19 trials should be designed to focus on specific patient groups that are more likely to benefit from the interventions that form the basis of the relevant clinical trials. Moreover, toward the goal of applying precision medicine to COVID-19 clinical practice, it should be possible to construct a quantifying domain of patient-profile signals (antecedent to trial commencement) to quantify the statistical probability that a given treatment will have a favorable patient outcome in relation to all the prior data in a patient's profile. Since researchers assume that certain factors in a patient's profile will be statistically correlated with favorable outcomes in conjunction with specific treatment plans, part of the trial's

[7] Reviewing the source code and documentation for CaPTk confirms that multi-application messaging along these lines is implicitly adopted by CaPTk; *see* for example https://www.med.upenn.edu/cbica/assets/user-content/images/captk/2018_ISBI_CaPTk.0404.Part2.pdf, particularly the material starting on the 30th slide of that presentation.

purpose is to determine which parts of the patient profile are, in fact, statistically relevant.

In practical terms, then, setting up COVID-19 trials would involve defining patient-selection criteria and implementing systems to screen for patients who may be good candidates for different trials. This would require two steps: (1) constructing a format where trial criteria can be rigorously notated; and (2) implementing software at participating sites to search for trial candidates. This software would need to access, represent, and analyze patient-specific trial-eligibility factors covering a broad spectrum of data types (sociodemographics, medical history, lab/image results, etc.). Proposing an automated recruitment engine for cancer trials, [29] argued (in 2014) that

> Many ... tools have been developed for accessing the institutional data warehouse to screen patients for clinical trials or for creating an alert system for physicians ... However, ... existing systems have limited access to the complete patient information, such as the latest laboratory test results, and are not integrated with the clinical systems used in routine patient care. Further, existing tools have limited support for structured entry of trial information, interactive user interfaces (UIs) that allow clinicians to review the matching results and re-execute the matching process with updates to patient records. [Although] there has been extensive research in creating formal, computable representation of eligibility trial specifications that can be used together with electronic representation of patient data in EHR systems [an] important challenge for computational representation of trial information is the lack of suitable interfaces for entering eligibility criteria. ***(page 2)***

Ten years earlier, [20] advocated for trial models "focused on interoperability, using modern object-oriented techniques" (page 3):

> Object-oriented software ... is more reliable because fewer software code changes are required as one's needs evolve; and, therefore, more changes can be made without effectively changing the core structure of the application ... The difference between object-oriented software and traditional software is substantial. ***(page 6)***

However, he concludes, "Very few CTMS products are object-oriented." Both [29] and [20], we should note, are not neutral observers, but writing to highlight features of software their own teams designed. Whatever the merits of their systems, these design principles have not apparently been incorporated into mainstream **CTMS** in intervening years.

Given the sheer scale of the SARS-CoV-2 pandemic, there are likely to be many candidates for almost any COVID-19 trial. However, methods for recruiting patients would need to be aligned across multiple institutions, at least in the case of multi-site trials. For example, in the context of antigen tests (measuring virus antibody levels), the US Centers for Disease Control recommends or has authorized a large list of assays performed by many different companies, using many different biochemical methods.[8] Because the data format resulting from immunoassays depends on the specific biochemical mechanism which (within each assay) yields quantitative data, a broad spectrum of antigen tests requires a diverse array of data formats that need to be integrated. As such, whenever COVID-19 immunoassays are considered as factors in immunological profiles for mapping patients to appropriate COVID-19 clinical trials, querying for good trial candidates means querying across a wide spectrum of structurally different data types that correspond to this broad array of antigen tests, specifically to the mechanisms through which laboratory instruments generate quantifiable data and to the computational procedures that process such data. Here is an example of why specialized trial software can be warranted: the heterogeneity of trial data, may call for integrative procedures implemented directly in programming languages such as **C++** (rather than query languages, such as **SQL**). The

[8] *See*, for example, https://www.fda.gov/medical-devices/coronavirus-disease-2019-covid-19-emergency-use-authorizations-medical-devices/vitro-diagnostics-euas#individual-antigen.

complexity of trial criteria was a motive for the software-based CTMS systems we cited earlier; COVID-19 serves as a trenchant case study supporting recommendations along those lines (*see also*, say, [17], [4], [33], [6], [38, especially pages 20–29]).

3.3.1 Customizing clinical trial management software

Once trials embrace heterogeneous data models (which require special-purpose software for accessing some of the trial data), Clinical Trial Management Systems (CTMS) requirements become more complex. In these situations, CTMSs may need to model and (sometimes) replicate complex computational workflows, such as those employed by Dearlove *et al.* for calculating SARS-CoV-2 genomic sequences from patients' blood samples. The CTMS software may also need to interoperate with domain-specific applications, as in bioimaging and image analyses, signal processing (e.g., for EKG analysis), flow cytometry, biochemical assays, genomic analysis, epidemiological modeling, and so forth. If possible, such applications should be configured or extended to work with the clinical trial software. For instance, if a DICOM (Digital Imaging and Communications in Medicine) client is used to study an image derived from a specific trial—e.g., a radiological scan of a COVID-19 patient's lungs—the DICOM software could be provided with a plugin that would show trial information in a separate window, which could then be juxtaposed with the main-image view. In this context, software alignment means that all institutions participating in a trial could use the *same* plugins, so that the trial's central CTMS system could interoperate with special-purpose software in a consistent manner. This would also aid in establishing, as part of the trial design, protocols for depositing special-purpose data assets (such as FCS or DICOM files) alongside clinical data and eCRFs.

A further benefit of CTMS customization is that custom software adds flexibility for trial design. By definition, trials allow researchers to test biomedical hypotheses in a controlled manner. Trials are therefore defined around the premise that observational information resulting from the trial is empirically significant, revealing something new about what the trial was designed to investigate. For instance, a COVID-19 trial might assess how well patients with varying prior immunological profiles respond to monoclonal antibody (mAB) treatments. The relevant observations in this case derive from the subsequent course of the disease for each patient, as well as potential adverse reactions, but there are inherent complicating factors: were patients receiving other treatments as well? For patients who recover, how do we know that the antibodies expedited that recovery? How quickly was the recovery? And, did patients continue to suffer from COVID-related symptoms even when they were no longer infectious? Situational details specific to the trial—such as each patient's mAB dosage level, prior COVID-19 risk factors, or viral-load change over time—also belong in the trial's unique data models. Moreover, a comprehensive investigation could well incorporate both information about the patient's unique immunological profiles and the nature of the SARS-CoV-2 variant/strain found in the patient.

In addition, customizing trial-management software has the added potential benefit of greater flexibility for incorporating personalized/patient-centered data into the overall trial results. Since patients' reactions to interventions are difficult to anticipate in full specificity ahead of time (especially when subjective experience is taken into account), allowing data models to evolve during the course of a trial can help trial designs respect the experiential dimensions intrinsic to patient-centered paradigms of care.

3.3.1.1 *Toward fine-grained sociodemographic models*

Patient-centric data could likewise include sociodemographic information about the patient, supplemented by epidemiological metrics, such as contact tracing. Designers need to identify, for example, what dimensions of patients' immunological and sociodemographic profiles are likely to be consequential when analyzing treatment outcomes (indeed, biomedical research has been criticized in recent years for bias toward certain populations, e.g., white, middle-class non-seniors). This results in uncertainties as to how well trial results carry over to populations at large; that is, populations characterized by a heterogeneous mix of demographic factors. Researchers can mitigate these concerns by demonstrating sociodemographic diversity among trial participants. Those goals, along with more precise predictive analytics, could be advanced by adopting more detailed sociodemographic reporting standards [18], [16], [21], etc.

Demonstrating sociodemographic diversity, however, calls for transparency about how sociodemographic details are represented. The process of grouping patients into ethnic/racial and/or socioeconomic strata can be equivocal at times. For example, if a trial participant is a graduate student at the University of Chicago, should their socioeconomic status be assessed on the basis of their own income or that of their parents? If their zip code places them on the South Side of Chicago, a region with both a prestigious campus and pockets of extreme poverty, should they be demographically classified alongside residents of that neighborhood? What about a varsity linebacker who appears to be in excellent health before a COVID-19 infection? Should his status as an athlete be taken to indicate being extremely fit prior to the disease, or might his background as a football player intimate a potential history of brain trauma, which may compound his neurological damage due to COVID-19?

In short, sociodemographic data can be notoriously imprecise. It is well-known that patients of lower socioeconomic status have higher COVID-19 infection rates and mortality rates than patients of higher socioeconomic status, but this disparity may be explained by several causal factors: more infectious workplaces, inferior post-infection treatment, poorer state of health before the onset of COVID, and so forth. Teasing apart these factors demands fine-grained analysis of the individual patient's pre-COVID history; sociodemographic generalizations can exclude important details. For example, for purposes of analysis, in the case of a graduate student with middle-class parents but with no health insurance of their own, how should we quantify their degree of access to health care? How well does geographic location serve as a proxy for socioeconomic status?

The case of a middle-class student living in a well-off near-campus corner of an otherwise impoverished neighborhood suggests that geospatial metrics (such as zip code or congressional district) are imperfect proxies for wealth; but other factors—such as air/water pollution or the risk of being the victim of a crime—may be statistically correlated among geographically proximate residents—even if they have otherwise divergent sociological profiles. These examples illustrate that the value of sociodemographic data is proportionate to the level of statistical detail with which the data may be analyzed. Instead of a broad and vague designation, such as "low income," one may want to derive more detailed subclasses, incorporating information about patients' employment, access to health care, physical fitness, and so forth. Two patients with similar income levels, for instance, may have different levels of access to health care (depending on factors such as whether the patients have employer-based insurance or geographic proximity to healthcare facilities), or different levels of exposure to COVID-19 in the workplace, depending on the nature of their job. Even if a trial does not quantify granular so-

ciodemographic assessments—such as, patients' workplace conditions and access to health care, which might serve as a more accurate estimation of how socioeconomic status causally affects disease outcomes—subsequent researchers may determine ways to analyze or add on to data generated by a trial (e.g., through follow-up studies of enrolled patients), so as to make such sociodemographic granularity part of the quantifiable framework.

3.3.1.2 Measuring cognitive and neurological effects

Similar interpretive issues also apply to post-treatment observations. How should researchers decide which observations qualify as clinically significant consequences of COVID-19? We have seen that as the pandemic has unfolded, a fair number of cases have been cited in the professional literature describing COVID-19 patients who suffer certain cognitive/neurological effects, such as muscular fatigue or weakness, mental confusion or poor concentration (sometimes referred to as "brain fog"), or symptoms of Guillain–Barré syndrome spectrum. Almost certainly, some of these symptoms may be the product of cognitive/neurological effects due to SARS-CoV-2. At the same time, COVID-19 patients—even those who fight off the infection successfully or who test positive, but remain asymptomatic—may find their lives so disrupted by the pandemic that this may (indirectly) cause cognitive and neurological problems. For instance, prolonged inactivity (for a typically active person), which commonly occurs as the result of a quarantine, may contribute to poor concentration and other diminished forms of mental acuity [11, say]. Given the fact that most people's lives during the pandemic are not "normal," it may be difficult to establish which symptoms experienced by a patient are actually biological effects of the disease itself or, alternatively, indirect consequences of lifestyle restrictions. This sort of ambiguity also applies to potential adverse side-effects of COVID-19 treatment. Rigorous design practice suggests, for instance, that parameters should be established framing the post-treatment and post-recovery window of time wherein patients' symptoms might be noted as potential effects of the disease or of the administered therapies themselves. How long after a treatment is administered should a patient's symptoms be considered potential side-effects of the therapy itself or, alternatively, the result of nagging uncertainties and a disrupted lifestyle imposed by the pandemic?

The fact that these questions have no predetermined answers indicates that trial designs need to anticipate a shifting clinical and information landscape as the trial evolves. For example, because SARS-CoV-2 was initially believed to affect lung functioning primarily, the risk of long-term cognitive/neurological damage was not widely anticipated when considering treatment over the course of COVID-19 infection. Consequently, because tests of a cognitive /neurological /physiological /radiographic nature (except for basic lung scans, with respect to radiology) had not been a common facet of early COVID-19 trials/observational studies, there were no corresponding data structures included in those studies for capturing this full spectrum of neuropsychological and neurological data.

Systematically tracking COVID-19's cognitive/neurological deficits entails neurological, laboratory, neuropsychological and radiographic tests to understand the full extent of their cognitive and neurological impairments: for example, the use of **MRI**s when considering COVID-related ischemic stroke [23], [13], the use of electrophysiological tests, cerebrospinal fluid tests (**CSF**), or the **MRC** (Medical Research Council) scale for muscle-strength test when considering COVID-related Guillain–Barré symptoms [27], or the use of neuropsychological tests, such as trail making test (**TMT**), sign coding test (**SCT**), continuous performance test (**CPT**), and digital span test (**DST**)—which measure a

patient's executive abilities (letter and number recognition mental flexibility, visual scanning, and motor function) and sustained and selective attention, along with other cognitive and neurological functioning—when considering cognitive/neuropsychological impairments after a serious bout with COVID-19 [41].

These cognitive, neurological, physiological, laboratory, and radiographic data structures thereby become an integral part of the information relevant to trial evaluation, because they document symptoms that are presumptively attributable to COVID-19. However, in practice, prior to clinicians having been alerted to the fact that COVID-19 may cause lingering cognitive/neurological damage, COVID-19 trials were not designed to incorporate neurological or cognitive data in a systematic manner. This scenario points to how trial data models can benefit from a built-in capacity to be redesigned while the trial is ongoing, so as to accommodate new and emerging information. On this basis, it is reasonable to adopt for clinical trials malleable information models, such as graph databases and/or object-oriented software components.

3.3.1.3 *Aggregating trial data via graph models*

A useful data-integration case study is the University of Pennsylvania's Carnival project (which achieves data integration by adopting property-graph databases, illustrating the flexibility of graph models in a way that we can also apply to trial design). Carnival synthesizes heterogeneous biomedical data by translating information from disparate sources into a common property-graph representation, and then querying this data with the Gremlin virtual machine. Gremlin is a "step-based" Virtual Machine, where "steps" between potential focus elements in a property graph play the role of primitive processing instructions; querying and traversing property graphs involves executing a series of Gremlin steps. Most Gremlin implementations are based on the Java programming language and the Java Virtual Machine (JVM), so that queries themselves are written in a JVM language (Groovy, in the case of Carnival). The challenge for any database engine that employs a relatively complex data-representation strategy, such as a hypergraph, property-graph, tuple-store (a collection of records with varying numbers of fields), or multi-dimensional (possibly sparse) array, is to efficiently map the high-level data structures manipulated within the database itself to the lower-level memory units that are stored to disk. Lower-level data structures are typically modeled via simpler database constructions, such as key-value stores, memory cells, or relational tables, so there must be a translation pipeline between high-level structures (properties, hypernodes, hyperedges, and so forth) and low-level points (record cells, shared memory address, elements in key-value pairs, etc.)

Consider how patient profiles usually notate the medications each patient is taking: any patient (a node) could, in principle, be connected to any medication (another node); some patients may be taking *no* medications, others may take just *one*, and some may take *two or more*. Also, connections between patients and medications can be the basis of further details that emerge over time (and are registered in the graph) inasmuch as medications are prescribed to patients by a specific doctor at a specific time, in a specific dosage, in response to specific diagnostic tests, and so forth. In short, information can "fan out" from the patient-to-medication connection in a relatively free-form manner. In general, then, as a subset of overall patient profiles, information about medication evinces the structural features, which are, in many contexts, optimally represented via free-form labeled graphs. Meanwhile, patient profiles may also consider medical history, which can be modeled as a graph with detailed logical and temporal internode connections. According to this representational strategy, each patient's history is a series of events and observations, which are tempo-

rally ordered—it is possible to query or traverse the graph in a manner that takes before/after relations into account—and where there are also logical or causal connections defined between nodes. For instance, an edge might assert that a given medication was prescribed to a patient (an event) *because of* the results of a given lab test (an observation). In these examples, different sorts of clinical data—sociodemographic, pharmacological, medical-history—are modeled according to different sorts of graph structures (hypernodes, nonschematic labeled edges, temporalized graphs, and so forth).

Insofar as different formations within data-structures tend to be conveyed via different graph-database constructions, the partition of graph-database technology itself into competing (partially incompatible) systems (distinct architectures, query languages, query-evaluation strategies, and so on) can also be a hindrance to data integration, even after adopting flexible graph-database technologies. This is one motivation for the hybrid graph models we discuss in later chapters, which attempt to recognize numerous structuring elements endemic to different flavors of graph systems, accommodated into a single hypergraph-based query framework.

3.3.2 Representing trial data via object models

As a concrete example of open-ended data-integration concerns being anticipated as a central element of trial design, consider how Object-Oriented models for COVID phylogeny (which we briefly considered last chapter) could serve as a nexus for integrating SARS-CoV-2 phylogenetic data across multiple studies and healthcare systems. Such an object model may be extended in different ways for different clinical trials, examining a range of COVID-19 treatments.

In the context of antibody regimens, for instance, scientists need to quantify how well the antibodies disable COVID-19 spike proteins directly, and/or how well the antibodies block the virus's ability to attach to human cells. These measurements generate data that gauge a patient's immune response to COVID-19, whether innate ("naive") or boosted by the administered antibodies. Researchers have mined such data from different angles, including contrasting symptoms presented with different levels of severity [22], [42], biochemically describing the mechanisms of immune response so as to augment or simulate that response via monoclonal antibodies or similar antiviral therapies [40], [7], identifying factors explaining at-risk groups' greater susceptibility to severe cases [25], or comparisons of SARS-CoV-2 against other coronaviruses [1] (these citations are representative examples of work conducted by many research groups; they could be expanded with similar references we included in Chapter 2 when discussing immune repertoires, for example). A COVID-19 software ecosystem should then ensure that such immune-response data can be effectively parsed and integrated into object models describing how SARS-CoV-2 is evolving around the globe, so researchers (as much as logistically feasible) could track each patients' immune response in light of their particular acquired SARS-CoV-2 strain/variant and their personal immunological profile. The relevant information for geospatial and sociodemographically tagged studies would then include patients' prior immuno-profiling and risk assessment, immune response (naive or affected by treatment) and clinical outcomes, as well as genetic tests for the viral variant. Cross-sectionally mapping such data geographically and along sociodemographic/socioeconomic contours could provide a holistic picture of the pandemic at a global scale.

Object models intended to facilitate such globally scoped data integration could then facilitate trials designed according to the protocol proposed by Shrestha *et al.*, discussed above, where each trial would study a preselected (non-random) cohort of patients for whom both

pre-treatment immunoprofiling data and post-treatment outcomes data would be available, so as to compare multiple trials against one another. Object models customized for each trial would generate a data framework through which the causal relations between patient profiles and treatment outcomes would be investigated. Customized trial software would, accordingly, provide a Reference Implementation demonstrating each trial-specific object model. If adopted for multiple trials conducted across multiple clinics/hospitals, trials' data models may help doctors better understand which aspects of patient profiles are particularly significant when matching COVID-19 patients to the most salutary treatment.

3.4 Text and data mining via CORD-19

The third methodological area we will concentrate on in this chapter is text and data mining, using analyses of large-scale document corpora and/or biomedical data sets to discover connections between research work, which might be less evident to individuals reading publications in isolation. Text mining and natural language processing (**NLP**) is certainly one methodology that has been explored for personalized/precision medicine, on the theory that automated searches across biomedical publication archives may detect diagnostically or prognostically significant connections between individual patients' profiles/symptoms and prior studies, which doctors might not discover otherwise. Overall, text-mining has been developed as one branch of data-mining and machine learning in general applied to document corpora under the aegis of precision-medicine (*see* for instance [5], [32], [31]).

Biomedical text mining is also a good case study in data-integration workflows, because typically the text-mining process requires synthesizing multiple **NLP** workflows and reading data from multiple input sources—[15], for instance, is a good precision-medicine context example—including sentence-parses, data sets of biomedical nomenclature, domain-specific knowledge bases (for gene-sequences, cancer variants, genomic-proteomic or genomic-antigen associations, and so forth), manual text annotations, etc.

Despite the perceived potential of patient-centered text mining, some scientists caution against overestimating the power of automated **NLP** platforms (*see* [14], for instance). These critiques are not rejecting text-mining in general, but rather observing limitations in existing document-encoding formats, which are derived from publishing technologies whose primary targets are human readers rather than machine-automated text processing. More systematic text representation and document annotation could alleviate the need for probabilistic **NLP** reasoning engines, making text-mining operations more precise and reliable.

As an example of the possibilities and challenges of text and data mining scenarios, we will consider **CORD-19**, a collection of COVID-19-related research articles that was developed (starting in Spring 2020) in conjunction with a White House "call to action" to spur COVID-19 research. This White House initiative was described as a "call to action ... to develop new text and data mining techniques that can help the science community answer high-priority scientific questions related to COVID".[9] As raw data for this initiative, the US Government helped spearhead a consortium of industry and academic institutions, headed by the Allen Institute for AI Research, who curated a "machine-readable Coronavirus literature collection" that includes article metadata and (in most cases) publication text for over 280,000 coronavirus research papers (as of mid-2021) [9], [39]. This corpus is paired with links to publisher portals (including

[9] *See* https://www.whitehouse.gov/briefings-statements/call-action-tech-community-new-machine-readable-covid-19-dataset.

Springer Nature, Wiley, Elsevier, the American Society for Microbiology, and the New England Journal of Medicine) providing full open access to COVID-19-related literature; these resources collectively constitute CORD-19 (the "COVID-19 Open Research Dataset").

3.4.1 Overview of CORD-19

The CORD-19 collection was formulated with the explicit goal of promoting both text mining and data mining solutions to advance coronavirus research. This means that CORD-19 is intended to be used both as a document archive for text mining and as a repository for finding and obtaining coronavirus data for subsequent research. The White House announcement directly requests institutions to develop *additional* technologies which would help scientists and jurisdictions to take advantage of CORD-19 as it was initially published. In short, CORD-19 was released with the explicit anticipation that industry and academia would augment the underlying data by layering on additional software.

Despite the obvious benefit to researchers, the health care community, and the public at large in publishers choosing to release a substantial quantity of COVID-19- related literature in open-access fashion, CORD-19 is not without certain limitations, as acknowledged by the curators themselves:

> We have performed some data cleaning that is sufficient to fuel most text mining & NLP research efforts. But we do not intend to provide sufficient cleaning for this data to be usable for directly consuming (reading) papers about COVID-19 or coronaviruses. There will always be some amount of error, which will make CORD-19 more/less usable for certain applications than others. We leave it up to the user to make this determination ...[10]

Some problems stem from how the articles are encoded into an ostensibly (JSON-based) machine-readable format, whereas others are unavoidable limitations of NLP overall.

To be clear, the concerns we identify here reflect an informal survey of CORD-19; they are not opinions provided by researchers working directly with CORD-19 or analyses discussed in peer-reviewed literature. With that caveat, however, we assert that certain issues deserve mention:

Transription Errors Transcription errors can cause the machine-readable text archive to misrepresent the structure and content of documents, hindering text-mining technology that targets the archive. In CORD-19, for instance, there are cases of scientific notation and terminology being improperly encoded. As a concrete example, "2'-C-ethynyl" is encoded in CORD-19 as "2 0-C-ethynyl", which could stymie text searches against the CORD-19 corpus (see [12] for the human-readable publication where this error is observed; the corresponding index in the corpus is 9555f44156bc5f2c6ac191dda2fb651501a7bd7b.json).

Poorly Indexed Research Data Although CORD-19 provides a structured representation of a large collection of research *papers*, there is no easy-to-use tool for finding research *data* through CORD-19.

Poorly Integrated Research Data The research data that *can* be accessed through CORD-19 evinces a wide variety of technical fields and formats, with distinct software requirements; as a result, it is a difficult task to merge and integrate different data sets related to COVID-19. At present, CORD-19 does not include any software tools or computer code that would facilitate data integration.

Disconnect Between Text Data and Publisher Portals Although most of the CORD-19 manuscripts represent peer-reviewed literature that can be accessed through document portals (for instance, the National Center for Biotechnology Information website), the CORD-19 archival schema does not represent these links (except indirectly via Document Object

[10] *See* https://github.com/allenai/cord19.

Identifiers). As such, there is no easy way for researchers to find and read publications which have been flagged by text-mining algorithms as being potentially of interest to them. Furthermore, there is no direct mechanism to enlarge the CORD-19 corpus with papers newly added to publisher portals.

To clarify the final comment: the Allen Institute for AI, which curated CORD-19, encourages publishers to contribute new (or newly available) articles to the corpus. However, integration with CORD-19 (or, so it seems, any other domain/topic-specific portal) is not developed as a formal step in the publication workflow. In particular, publishers are not themselves generating machine-readable document infosets that can be integrated with the CORD-19 schema (which, in turn, causes transcriptions errors and other problems as just outlined).

With respect to text mining, an immediate problem arises in CORD-19's archive-construction methodology: especially, how the text was parsed from PDF files. This is a process which almost inevitably causes imprecise or inaccurate text representation, which can degrade the quality of the archive, unless manual or automated corrections are made. In particular, the CORD-19 library evinces transcription errors, as mentioned above (especially in relation to technical or scientific phrases and terminology); scientific notation in particular may be improperly encoded. Moreover, there is no semantic marking identifying that (say) the "2 0-C-ethynyl" text segment has a specific technical meaning. These errors or limitations arise in part from unavoidable anomalies which occur when reading texts from PDF files, rather than from machine-readable, structured formats such as XML.

It is also worth observing that the JSON format used for encoding CORD-19 manuscripts presents some logistical challenges for any operations related to text-mining or to cross-referencing publications and data sets. In particular, CORD-19 makes partial use of "standoff annotation"; specifically, document features, such as citations and references, are notated through character offsets into the paragraph where they appear. As a result, accurately reading these document elements requires synthesizing data points parsed from several distinct objects in the JSON code, which is only feasible given a client library built to interface with the CORD-19 files in accord with their specific schema. Such a client library would implement convenience procedures to handle recurring tasks, such as obtaining the full bibliographic reference affixed to a given location in a manuscript.

With respect to *data* mining in the CORD-19 context, the limitations in the currently available raw CORD-19 data are even more pronounced than in the context of text mining. In particular, neither the article metadata nor the full open-access document collections have any mechanism for actually obtaining data published alongside research papers, or even identifying which papers have accompanying data in the first place. The Springer Nature collection which was originally one component within CORD-19 illustrates the limitations of this relatively unstructured data-publishing approach (this following analysis will focus on Springer Nature, but the problems identified are no less pronounced on the other CORD-19 portals; if anything, because Springer Nature allows readers to browse articles in HTML within the web portal directly, one can ascertain whether research data exists for an article without downloading and reading it; with other CORD-19 resources it is actually harder to locate supplemental data when available). Initially, the Springer Nature portal encompassed 43 articles, of which 15 were accompanied by research data that could be separately downloaded (this number does not include papers that document research findings only indirectly, via tables or graphics printed inline with the text). Collectively these articles referenced over 30 distinct data sets (some papers were linked to multiple data sets), forming a data collection which could be a valuable resource for COVID-19 research, not only through

the raw data made available, but as a kernel around which new coronavirus data could accumulate. However, there is currently no mechanism to make this overall collection available as a single resource.[11]

This problem demonstrates, among other things, how document-metadata formats such as the research object protocol are limited in applying only to *single* articles. As a result, there is no commensurate protocol for publishing *groups* of articles which are tied to groups of data sets unified into an integral whole. Open-access COVID-19 papers also reveal limitations of exiting online document portals, especially with respect to how publications are linked to data sets. In particular, there is no clear indication that a given paper is associated with downloadable data; usually readers ascertain this information only by reading or scrolling down to a "supplemental materials" or "data availability" addendum near the end of the article. Moreover, because the Springer Nature portal (and similar publisher resources) aggregates papers from multiple sources, there is no consistent pattern for locating data sets; each journal or publisher has their own mechanism for alerting readers to the existence of open-access data and citing where they could be downloaded.

3.4.2 Data integration within CORD-19

Aside from the issues that are likely to hinder text and data mining across **CORD-19**, the collective group of COVID-19 data sets also illustrates the limitations of information spaces pieced together from disconnected raw data files with little additional curation. The files included in this group of data sets encompass a wide array of file types and formats, including **FASTA** (which stands for Fast-All, a genomics format), **SRA** (sequence read archive, for **DNA** sequencing), **PDB** (Protein Data Bank, representing the **3D** geometry of protein molecules), **MAP** (electron microscopy map), **EPS** (Embedded Postscript), and **CSV** (comma-separated values). There are also tables represented in Microsoft Word or Excel formats. Although these various formats are reasonable for storing raw data, not all of them are actually machine-readable; in particular, the **EPS**, Word, and Excel files need manual processing to use the information they provide in a computational manner. A properly curated data collection would need to unify disparate sources into a common machine-readable representation (such as **XML**).

Furthermore, productive data curation should also aspire to *semantic* integration, unifying disparate sources into a common data model. For example, multiple spreadsheets among the Springer Nature COVID-19 data sets hold sociodemographic and epidemiological information relevant to modeling the spread of the disease. These different sources could certainly be integrated into a canonical social-epidemiology-based representational paradigm which recognizes the disparate data points that might be relevant for tracking the spread of COVID-19 (with the potential to unify data from many countries and jurisdictions).

This is not only an issue of data *representation* (viz., how data is physically laid out), but also of data types and computer code. According to the Research Object protocol, data sets should include a code base, which provides convenient computational access to the published data. In the case of COVID-19, this entails creating a sociodemographic and epidemiological code library optimized for COVID-19 information, which would be the primary access point for researchers seeking to use the data that has been published in conjunction with the

[11] As **CORD-19** has evolved, the publisher-specific sections therein appear to be merged into portals such as Springer Nature directly, so our above comments based on isolating Springer Nature articles are probably more applicable to the original archive design than the current technology. However, insofar as the current portal simply defers to publisher-specific search features, we would argue that accessing COVID-19 data sets through **CORD-19** is if anything more difficult than before.

43 manuscripts examined here, which were aggregated on Springer Nature, along with any other coronavirus research that comes online. Similar comments apply not only to tabular data represented in spreadsheet or **CSV** form, but to more complex molecular or microscopy data that needs specialized scientific software to be properly visualized.

Considering the overall space of COVID-19 data, it is unavoidable that some files require special applications and cannot be directly merged with the overall collection. For instance, there is no coherent semantics for unifying Protein Data Bank files with sociodemographics and epidemiology; files of the former type have specific scientific uses and can only be understood by special-purpose software. Nevertheless, a downloadable COVID-19 archive can include source code for code libraries reading special formats, such as **PDB** or **FCS** (as a case study, this book's supplemental materials provides a build-environment for tools working with those file types, among others).

Earlier we advocated for Object-Oriented models of COVID-19 variants that could be integrated with both chemical/molecular data and sociodemographic/epidemiological data. Different COVID-19 strains would then form a bridge linking these different sorts of information; researchers should be able to pass back and forth from molecular or genomic visualizations of COVID-19 to social-epidemiological charts and tables based on viral strains. Ideally, capabilities for this sort of interdisciplinary data integration would be provided by a COVID-19 archive as enhancements to applications that scientists would use to study the published data.

It is worth noting that a data-mining platform requires *machine-readable* open-access research data, which is a more stringent requirement than simply publishing data alongside that can be understood by domain-specific software. For example, radiological imaging can be a source of COVID-19 data insofar as patterns of lung scarring, such as "ground-glass opacity," is a leading indicator of the disease. Consequently, diagnostic images of COVID-19 patients are a relevant kind of content for inclusion in a COVID-19 data set (see [30] as a case study). However, diagnostic images are not in themselves "machine readable." When medical imaging is used in a quantitative context (e.g., applying machine learning for diagnostic pathology), it is necessary to perform image analysis to convert the raw data (viz., in this case, radiological graphics) into quantitative aggregates (for instance by using image segmentation to demarcate geometric boundaries, and then defining diagnostically relevant features, such as opacity, as a scalar field over the segments). In short, even after research data is openly published by article authors, it may be necessary to perform additional analysis on the data for it to be a full-fledged component of a machine-readable information space.[12]

Another concern in developing an integrated **CORD-19** data collection is that, logistically speaking, not all COVID-19 data is practical to reuse as a downloadable package. This is especially true for genomics; several of the aforementioned 43 coronavirus papers included data published via online data banks capable of hosting data sets that are too large for an ordinary computer. In these situations scientists formulate queries or analytic scripts that are sent remotely to the online repositories, so that researchers access the actual published data only indirectly. Nevertheless, access to these data sets can still be curated as part of a COVID-19 package; in particular, computer code can be provided which automates the process of networking with remote genomics archives through the accession numbers and file-formats which those archives recognize.

[12] This does not mean that diagnostic images (or other graphical data) should be excluded from data sets; only that computational reuse of such data will usually involve certain numeric processing, such as image segmentation. Insofar as this subsequent analysis is performed, the resulting data can be added as a supplement to the image data set itself.

As a final point on the topic of integrating disparate **CORD-19** research data, note that an overarching framework for indexing COVID-19 data sets would also facilitate the process of cross-referencing article text and research data. In particular, the annotation system employed for **CORD-19** could profitably be enhanced by a system of *microcitations* that apply to portions of manuscripts *as well as* data sets. In the publishing context, a microcitation is defined as a reference to a partially isolated fragment of a larger document, such as a table or figure illustration, or a sentence or paragraph defining a technical term, or (in mathematics) the statement/proof of a definition, axiom, or theorem. In data publishing, "data citations" are unique references to data sets in their entirety, or to smaller parts of data sets. A data microcitation is then a fine-grained reference into a data set: for example, "the precise data records actually used in a study" (as defined by the Federation of Earth Science Information Partners; see [24]), one column in a spreadsheet, or one statistical parameter in a quantitative analysis.

Ideally, the text-mining and data-mining notions of microcitation should be combined into a unified framework. In particular, text-based searches against the **CORD-19** corpus should also try to find matches in the data sets accompanying articles within the corpus. As a concrete example, a concept such as "expiratory flow" appears in **CORD-19** both as a table column in research data, and as a medical concept discussed in research papers; a unified microcitation framework should therefore map *expiratory flow* as a keyphrase to both textual locations and data set parameters. Similarly, a concept such as *2′-C-ethynyl* (mentioned earlier in the context of transcription errors) should be identified both as a phrase in article texts and as a molecular component present within compounds whose scientific properties are investigated through **CORD-19** research data, so that a search for this concept can trigger both publication and dataset matches. Implementing this kind of unified search mechanism requires that data sets be *annotated* with techniques similar to those used for marking up natural language techniques.

Considering the inter-disciplinary nature of COVID-19 research, it is unavoidable that different scientists will need different sorts of specialized software to analyze the kinds of information relevant to their research. For instance, the computational techniques applicable to diagnosing coronavirus infection are scientifically very different from those used for genomic or epidemiological studies of the disease; pathologists would not in general use the same software as for genomics/bioinformatics, or virology/epidemiology. In short, even while scientists start with a common pool of raw data, they will need to analyze this data through a diverse set of supplemental computational tools, which will vary not only across disciplines, but also in terms of the software and laboratory facilities available to different researchers through their institutions.

In the next chapter, we will argue that biomedical research corpora have some similarities to "data lakes" and similar large-scale, heterogeneous information systems maintained by hospitals and other clinical and/or research institutions. These dynamics arguably extend beyond research *data* to include document archives such as **CORD-19** as well. Such archives (with **CORD-19** as a case in point) would be centralized in the sense of employing a single curation, accession, and data-management protocol, but would branch out into many distinct research areas, corresponding to data being consumed and studied through a wide range of software products and ecosystems. Common functionality for basic data-acquisition capabilities would then need to be shared among a spectrum of software components which are otherwise variegated in terms of the data formats and computational resources they can provide or recognize. Ideally, this mixture of feature alignment and diversification would be anticipated in the design of document corpora and corresponding protocols

for accessing data sets associated with included publications, where applicable.

The prior paragraphs have highlighted limitations of data sets published in conjunction with coronavirus articles made available as open-access resources on Springer Nature (and, by extension, CORD-19). The central point here is to argue for a distinct data-curation stage in the publication process, with data curators playing a role distinct from that of both authors and editors.[13] Moreover, the discussion has hopefully highlighted problems with current data-sharing paradigms, even those such as the Research Object and FAIR initiatives that are explicitly devoted to improving how open-access data sets are published. CORD-19 exposes several lacunae in the Research Object protocol, for example, which point to the need for a more detailed extension of this protocol. In particular, an enhanced protocol should encompass (*inter alia*) the following:

1. A canonical framework for archiving collections of data sets, not only single data sets (and not only groups of data sets published with a single research paper). For example, all data sets published alongside the 43 Springer Nature articles could be unified into a single collection.
2. A code base accompanying data-set collections designed to help research unify the information provided. Curating the overall collection would involve pooling disparate data into common representation, and implementing computer code that deserializes and processes the unified data accordingly. For instance, CSV, EPS, and Microsoft Word/Excel tables could be migrated to XML, JSON, or a more complex common format. Customized computer code could then be implemented specifically to parse and merge the information present in single data sets within the overall collection. This implementation would reciprocate the Research Object goal of unifying code and data, but, again, at the level of an aggregate of research projects, rather than a single Research Object.
3. A unified data-set collection should be self-contained as much as possible, and should be built around a foundation where advanced computing capabilities are available in a transparent, standalone fashion, without requiring tools outside the collection itself. One way to achieve this is by gravitating toward components that can provide features such as scripting and data persistence through components that can be shared in pure source-code fashion, such as the WhiteDB database engine [28] and the AngelScript scripting language.[14]
4. A unified data-set collection should also provide prototyping and remote-access tools to interface with web-based information spaces that host data sets too large to be individually downloaded. Ideally, these would include simulations of remote services, which would help scientists understand the design of the remote archives and how to interface with them.
5. Finally, a unified research portal could influence the design of web portals where associated texts are published (*see* [8], [2], [34], [19], for instance). Ideally, it would be easy for readers to identify which articles have supplemental data files and to download those files if desired. Moreover, a microcitation framework could be made available for textual links between publication content and data sets; for instance, a plot or diagram illustrating statistical or functional distributions

[13] The point here is not to critique the work of individual authors; curating data sets according to exacting scientific standards demands a vein of expertise that typically lies outside researchers' disciplinary scope. The point is rather that publishers should recognize data curation as a distinct process and skill-set complementary to both writing and editing research works.

[14] *See* http://www.angelcode.com/angelscript.

should link to the portion of the data set from which that quantitative data is derived.

This discussion has used the COVID-19 crisis as a lens through which to examine data-publishing limitations in general. Such limitations are not specific to coronavirus in particular. However, the nearly unprecedented urgency of this epidemic reveals how both the scientific and publishing industries are still struggling to develop technologies and practices which keep pace with the intersecting needs of systematic research and public policy. An optimistic projection is that the crisis will spur momentum toward a more sophisticated data-sharing paradigm—perhaps a generalization of the Research Object protocol toward data-set collections.

3.4.3 Reviewing the CORD-19 document model

To discuss the possibilities and limitations of CORD-19 (and potentially other document corpora with a similar design), it is worth examining how CORD-19 encodes textual data in greater detail. This discussion has ramifications outside of CORD-19 itself, insofar as CORD-19 hopefully points to gaps in current publishing technologies. These gaps need to be addressed if publishers are to curate open-access corpora that truly leverage the digital and interactive technology available to us with modern software.

The basis of CORD-19's infrastructure is a JSON scheme that describes the document hierarchy of research articles encoded within the corpus. Apart from metadata (consisting of basic details, such as document title and authors' names) and bibliographic entries, all document content according to this schema is divided into paragraphs (implicitly the documents are divided into sections as well, but sections are notated as properties of the paragraphs they contain, not as a separate level in the hierarchy). Each paragraph encoding contains an underlying string vector (a stream of characters) and, separate and apart from that, character "spans" which point to references (such as Named Entities), citations, and equations. This indicates that the CORD-19 encoding uses "standoff annotation," where any content modifying the interpretation assigned to portions of the main text is notated with a series of data structures described apart from the main text itself.

Standoff text-encoding systems may be contrasted with XML or HTML, where "tags" are mixed with character data. For example, consider a span of text that quotes from another document: in HTML, the special status of the quoted text may be marked by surrounding the text with **<quote>** start and end tags. Syntactically, this markup system has the effect that tags and text are seen side-by-side: any content governed by the **<quote>** (i.e., the text of the quote itself) is printed immediately after the begin-tag, and the quotation ends when the last character is followed by an end-tag (i.e., **</quote>**).

Apart from such syntactic details, the distinction between tag-based markup and standoff annotation determines the "semantics" of the document, insofar as tags form a document hierarchy. Continuing the **<quote>** example, the text-span inside the quote tags is represented as a *child element* of the quote, whereas the quote itself may be a child element of a larger-scale entity (such as a paragraph). In effect, the paragraph *contains* a quote, and the quote *contains* a string of characters. Such nested levels of containment provide the structure through which hierarchical documents (formats such as XML and HTML) are interpreted.

To see the contrast with *standoff* annotation, if one were to describe a document using a standoff annotation system, the notation that a particular span of characters belongs to a quotation would not be marked-up amidst the characters themselves. Instead, the quotation-designation would use numeric indices to declare that the character at a certain position in the main text

begins a quotation, and some later character in the text ends that quotation. When serializing documents with standoff annotation, all the characters in a document are typically represented as one character-stream, and any notation describing markup applied to spans within that character stream is asserted afterward, using indexes into the stream to demarcate element boundaries.

The JSON schema used for CORD-19 is not entirely standoff, because there is a document hierarchy (for example, a publication's abstract is modeled as a sibling element to the main body text, so abstracts and the main text represent an intermediate hierarchical level, contained within the overall document and containing individual paragraphs). However, CORD-19 uses, in effect, a standoff-annotation system for each paragraph, so there is no hierarchical level smaller than paragraphs themselves, except implicitly. After the text (viz., the character stream), there are subsequent notations of spans within the paragraph (each span description is considered a child of the paragraph itself, as is the paragraph text).

This arrangement has consequences for text mining algorithms, which may be strengths or weaknesses in different contexts. One consequence is that the raw text is all grouped together in one place; algorithms do not have to tie together child nodes of disparate XML elements to derive a beginning-to-end sequence of the text belonging to any paragraph. Instead, it is simply necessary to read all data in the "text" field of the relevant "paragraph" object. The character-sequence in this text may contain words and sentences, but, potentially, other strings of symbols (such as chemical formulae), which are not explicitly marked. This may or may not be desirable. It could potentially complicate NLP tasks, because the language-processing components will be fed not only sequences of English words, but also, sometimes interspersed among ordinary words, technical symbol-sequences, such as "2'-C-ethynyl" (an example used earlier in this chapter). Standoff annotations may or may not be effective in marking the boundaries of such extra-lexical sequences; certainly we cannot rely on Named Entity detectors to properly identify and demarcate the boundaries of all uses of technical terminology or special symbols (again, the limitations of automated annotation are discussed earlier in this chapter).

In discussing standoff annotation, it is also worth considering how the text of CORD-19 publications was obtained. According to CORD-19 documentation, most full-text transcriptions in the corpus were obtained from PDF files, via a pipeline using TEI (text encoding initiative) XML as an intermediate representation. Necessarily, then the encoded text is only an approximate representation of the original:

> To provide accessible and canonical structured full text, we parse content from PDFs and associated paper documents. The full text is presented in a JSON schema designed to preserve most relevant paper structures such as paragraph breaks, section headers, and inline references and citations. ... We recognize that converting between PDF or XML to JSON is lossy. However, the benefits of a standard structured format, and the ability to reuse and share annotations made on top of that format have been critical to the success of CORD-19. ... Though we have made the structured full text of many scientific papers available to researchers through CORD-19, a number of challenges prevent easy application of NLP and text mining techniques to these papers. First, the primary distribution format of scientific papers—PDF—is not amenable to text processing. The PDF file format is designed to share electronic documents rendered faithfully for reading and printing, not for automated analysis of document content. Paper content (text, images, bibliography) and metadata extracted from PDF are imperfect and require significant cleaning before they can be used for analysis. Second, there is a clear need for more scientific content to be made easily accessible to researchers. Though many publishers have generously made COVID-19 papers available during this time, there are still bottlenecks to information access. ... Lastly, there is no standard format for representing paper metadata. Existing schemas like ... JATS[,] Crossref [or] Dublin Core have been adopted as representations for paper metadata. However, there are issues with these standards; they can

> be too coarse-grained to capture all necessary paper metadata elements, or lack a strict schema. ... Without solutions to the above problems, NLP on COVID-19 research and scientific research in general will remain difficult. ***[39, page 6]***

As an example of these NLP issues, consider the challenge of demarcating all named entities, particularly technical character-sequences (such as chemical formulae) which are not ordinary lexemes. Whether or not authors explicitly mark up such sequences (they may well do so in that formulae or equations are often typeset differently than normal text) this markup is not preserved in PDF versions of articles. As the authors of the last-cited article point out, many (roughly 38%) of papers in CORD-19 are also available in the JATS (journal article tag suite) format, which is a more precise text encoding than PDF. However, even in this context JATS does not compel authors to explicitly notate textual entities, such as special terms or character-sequences—in fact, JATS does not truly have an obvious structure or set of alternative structures for identifying what would normally be considered annotation-worthy text spans or named entities; the closest correlates are probably the generic **<kwd>** (keyword) and **<abbrev>** (abbreviation) tags as well as discipline-specific options, such as **<chem-struct>** (for chemical structures) and **<disp-formula>** (for mathematical expressions). In short, building a corpus such as CORD-19 for rigorous text-mining is made more difficult because authors and publishers do not publish texts in formats which are optimized for text mining in the first place; the acknowledged limitations of CORD-19 reflect problems of industry practice, not programming lacunae that could be alleviated with more sophisticated NLP algorithms.

Having acknowledged these limitations, a discussion of document corpora could then reasonably pivot from the empirical goal of curating useful text archives from currently published text to examining how more sophisticated corpora may be published in the future. It is reasonable, for example, to propose that full-text publications be released *both* in reader-friendly PDF form *and* in machine-readable forms, such as JATS. This is not just an abstract proposal; indeed, the text of this very book has been prepared using a novel document-generation system that creates both machine-readable structured text and PDF output, moreover with cross-referencing between them; notably, the positions of discursively important textual markers, such as sentence boundaries, are mapped to PDF screen coordinates (the code library for the book includes document-generation code as well as the data set of coordinate positions generated as part of the book's publication workflow). In particular, it is reasonable for authors and editors to manually introduce textual annotations for content such as named entities, keywords, important technical terms, and other content which should be targeted by NLP engines separate and apart from ordinary lexemes with their conventional natural-language semantics. Typically such specialized terms/lexemes would be marked up in any case, because they may require distinct fonts or styling than their surroundings. It is also reasonable to manually define sentence boundaries via simple rules (e.g., two following spaces mark the end of a sentence; a single space, such as that following an abbreviation, indicates situations where a character such as a period, which could potentially mark the end of a sentence, is actually playing a different discursive role).

By following simple rules of document content-entry and lexicography, certain NLP tasks, such as sentence-boundary and Named Entity recognition, can be optimized—eliminating the need for probabilistic algorithms and relying instead on much less sophisticated, but more accurate, markup-parsing logic. If sentence boundaries and named entities are explicitly annotated in machine-readable text encodings, then extracting these features is not really an issue of "natural language processing" as such. On the

other hand, AI-driven analysis of document corpora would still require NLP for other aspects of parsing documents; it is unreasonable to expect authors, for instance, to manually notate sentence parse-graphs. This then suggests the question of where the boundary lies between discursive structures which might reasonably be left to authors or editors to manually notate (e.g., sentence boundaries) and those which in practice could only be obtained via NLP (such as part-of-speech tags). Related to this question is how best to model NLP structures, such as the trees or graphs representing the syntax of natural-language sentences. We will consider this question in subsequent chapters in the context of Conceptual Space Theory.

References

[1] Teresa Aydillo, et al., Immunological imprinting of the antibody response in COVID-19 patients, Nature Communications 12 (2021), https://www.nature.com/articles/s41467-021-23977-1.

[2] Alessia Bardi, et al., Enhanced Publication Management Systems: A Systemic Approach Towards Modern Scientific Communication, Dissertation, University of Pisa, 2016, https://core.ac.uk/download/pdf/79621484.pdf.

[3] Khalid Belhajjame, et al., Workflow-centric research objects: first class citizens in scholarly discourse, https://users.ox.ac.uk/~oerc0033/preprints/sepublica2012.pdf.

[4] Crenguta Bogdan, et al., Towards a Clinical Trial Ontology Using a Concern-Oriented-Approach, IRPPS Working Papers, vol. 10, 2006, https://www.movetothecloud.it/irpps/e-pub/index.php/wp/article/view/10/78.

[5] Kevin Bretonnel Cohen, Lawrence E. Hunter, Text mining for translational bioinformatics, PLoS Computational Biology 9 (4) (2013), https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003044.

[6] Steve Canham, Christian Ohmann, A metadata schema for data objects in clinical research, Trials 17 (2016), https://trialsjournal.biomedcentral.com/articles/10.1186/s13063-016-1686-5.

[7] Yao-Qing Chen, et al., Rapid isolation and immune profiling of SARS-CoV-2 specific memory B cell in convalescent COVID-19 patients via LIBRA-seq, Signal Transduction and Targeted Therapy 6 (2021), https://www.nature.com/articles/s41392-021-00610-7.

[8] Helena Cousijn, et al., A data citation roadmap for scientific publishers, Nature Scientific Data 5 (2018), https://www.nature.com/articles/sdata2018259.pdf.

[9] COVID-19 open research dataset (CORD-19), version 2020-03-13. Retrieved from https://pages.semanticscholar.org/coronavirus-research, 2020. (Accessed 20 March 2020), https://doi.++org/10.5281/zenodo.3715506, https://pages.semanticscholar.org/coronavirus-research.

[10] Christos Davatzikos, et al., Cancer imaging phenomics toolkit: quantitative imaging analytics for precision diagnostics and predictive modeling of clinical outcome, Journal of Medical Imaging 5 (1) (2018), https://www.nmr.mgh.harvard.edu/~you2/publications/Davatzikos18_CaPTk.pdf.

[11] Sonia Difrancesco, et al., Sociodemographic, health and lifestyle, sampling, and mental health determinants of 24-hour motor activity patterns: observational study, Journal of Medical Internet Research 23 (2) (2021), https://www.jmir.org/2021/2/e20700.

[12] Ludëk Eyer, et al., Nucleoside analogs as a rich source of antiviral agents active against arthropod-borne flaviviruses, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5890575.

[13] Johanna T. Fifi, J. Mocco, COVID-19 related stroke in young individuals, The Lancet 19 (2020), https://www.thelancet.com/journals/laneur/article/PIIS1474-4422(20)30272-6/fulltext.

[14] Holger Fröhlich, et al., From hype to reality: data science enabling personalized medicine, BMC Medicine 16 (2018), https://bmcmedicine.biomedcentral.com/articles/10.1186/s12916-018-1122-7.

[15] Obi L. Griffith, et al., Text-mining clinically relevant cancer biomarkers for curation into the CIViC database, Genome Medicine 11 (2019), https://genomemedicine.biomedcentral.com/articles/10.1186/s13073-019-0686-y.

[16] Maritt Kirst, et al., Sociodemographic data collection for health equity measurement: a mixed methods study examining public opinions, International Journal for Equity in Health 12 (2013), https://equityhealthj.biomedcentral.com/articles/10.1186/1475-9276-12-75.

[17] Li-Min Liu, et al., A new software development methodology for clinical trial systems, Advances in Software Engineering 2013 (2013), https://www.hindawi.com/journals/ase/2013/796505.

[18] Aaron M. Orkin, et al., Reporting of sociodemographic variables in randomized clinical trials, JAMA Network Open 4 (6) (2021), https://jamanetwork.com/journals/jamanetworkopen/fullarticle/2780555.

[19] Barbara McGillivray, et al., The citation advantage of linking publications to research data, PLoS ONE (2020), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0230416.

[20] John S. McIlwain, Clinical trial management systems (CTMS) system selection considerations, https://velos.com/wp-content/uploads/Clinical-Trial-Management-Systems-System-Selection-Considerations-Whitepaper-1.pdf.
[21] Shelby Meyer, et al., Sociodemographic diversity in cancer clinical trials: new findings on the effect of race and ethnicity, Contemporary Clinical Trials Communications 21 (2021), https://www.sciencedirect.com/science/article/pii/S245186542100020X.
[22] Zachary Montague, et al., Dynamics of B cell repertoires and emergence of cross-reactive responses in patients with different severities of COVID-19, Cell Reports 35 (2021), https://www.sciencedirect.com/science/article/pii/S2211124721005180.
[23] Ross W. Paterson, et al., The emerging spectrum of COVID-19 neurology: clinical, radiological and laboratory findings, Brain 143 (10) (2020) 3104–3120, https://academic.oup.com/brain/advance-article/doi/10.1093/brain/awaa240/5868408.
[24] Mark A. Parsons, Ruth Duerr, Data identifiers, versioning, and micro-citation, https://www.thelancet.com/action/showPdf?pii=S1473-3099%2820%2930086-4.
[25] Lisa Paschold, et al., SARS-CoV-2-specific antibody rearrangements in prepandemic immune repertoires of risk cohorts and patients with COVID-19, Journal of Clinical Investigation (2020), https://www.jci.org/articles/view/142966.
[26] Philip R.O. Payne, Translational informatics: enabling high-throughput research paradigms, Physiology Genomics 39 (2009) 131–140, https://journals.physiology.org/doi/pdf/10.1152/physiolgenomics.00050.2009.
[27] Samir Abu-Rumeileh, et al., Guillain-Barré syndrome spectrum associated with COVID-19: an up-to-date systematic review of 73 cases, Journal of Neurology 268 (4) (2020) 1133–1170, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7445716.
[28] Enar Reilent, Whiteboard architecture for the multi-agent sensor systems, https://www.thelancet.com/action/showPdf?pii=S1473-3099%2820%2930086-4.
[29] Sayta S. Sahoo, et al., Trial prospector: matching patients with cancer research studies using an automated and scalable approach, Cancer Informatics 13 (2014) 157–166, https://pubmed.ncbi.nlm.nih.gov/25506198.
[30] Heshui Shi, et al., Radiological findings from 81 patients with COVID-19 pneumonia in Wuhan, China: a descriptive study, https://www.thelancet.com/action/showPdf?pii=S1473-3099%2820%2930086-4.
[31] Michael Simmons, et al., Text mining for precision medicine: bringing structure to EHRs and biomedical literature to understand genes and health, Advances in Experimental Medicine and Biology (2016) 139–166, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5931382.
[32] Ayush Singhal, et al., Text mining for precision medicine: automating disease-mutation relationship extraction from biomedical literature, Journal of the American Medical Informatics Association 23 (4) (2016) 766–772, https://academic.oup.com/jamia/article/23/4/766/2201020.
[33] Michael Souillard, et al., A flexible framework for managing temporal clinical trial data, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.75.9238.
[34] Markus Suhr, et al., Menoci: lightweight extensible web portal enabling FAIR data management for biomedical research projects, BMC Bioinformatics 21 (2020), https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-03928-1.
[35] Gentle Sunder Shrestha, et al., Precision medicine for COVID-19: a call for better clinical trials, Critical Care 24 (2020), https://ccforum.biomedcentral.com/articles/10.1186/s13054-020-03002-5.
[36] Alina Trifan, José Luís Oliveira, FAIRness in biomedical data discovery, in: 12th International Conference on Health Informatics, Proceedings, 2019, pp. 159–166, https://www.scitepress.org/Papers/2019/75764/75764.pdf.
[37] Johan Van Soest, et al., Towards a semantic PACS: using semantic web technology to represent imaging data, Studies in Health Technology and Informatics 205 (2014) 166–179, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5119276.
[38] Kris Verlaenen, Middleware for Advanced Service Configuration: a Policy-Based Approach, Dissertation, Leuven, 2008, https://www.cs.kuleuven.be/publicaties/doctoraten/cw/CW2008_07.pdf.
[39] Lucy Lu Wang, et al., CORD-19: the COVID-19 open research dataset, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7251955, 2020.
[40] Pingping Wang, et al., Comprehensive analysis of TCR repertoire in COVID-19 using single cell sequencing, Genomics 113 (2) (2021) 456–462, https://www.sciencedirect.com/science/article/abs/pii/S0888754320320838.
[41] Hetong Zhou, et al., The landscape of cognitive function in recovered COVID-19 patients, Journal of Psychiatric Research 128 (2020) 98–102, https://www.sciencedirect.com/science/article/abs/pii/S0022395620308542.
[42] Yonggang Zhou, et al., Profiling of the immune repertoire in COVID-19 patients with mild, severe, convalescent, or retesting-positive status, Journal of Autoimmunity 118 (2021), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7837046.
[43] Monica L. Zigman Suchsland, et al., Patient-centered outcomes related to imaging testing in US primary care, Journal of the American College of Radiology 16 (2) (2019) 156–163, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7050575.

This page intentionally left blank

CHAPTER

4

Modular design, image biomarkers, and radiomics

OUTLINE

4.1 Introduction

In this chapter, and again in Chapter 8, we will consider image-annotations as a case study in the methodology we will describe as "multi-aspect modular" design. Modular design in general attempts to realize larger-scale projects by subdividing them into relatively autonomous smaller parts. We propose the term "multi-aspect modules" to describe software components which are not monolithic applications—they are intended to be mixed with other modules to create full applications—but which offer integrated functionality in a self-contained fashion. For example, each module would autonomously implement GUI classes, database access, data serialization, and scripting/runtime-reflection features.

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00009-X

Image-annotations furnish a good example of how concerns related to GUIs, serialization, data persistence, and procedural models are interconnected. Bioimage annotations are also a useful starting-point from which to consider how multi-aspect design might be adopted for other branches of bioinformatics and biomedical software engineering, because bioimaging data is, intrinsically, conceptually linked to other kinds of clinical/diagnostic information. We assume, then, that hypothetical image-annotation modules are deployed in conjunction with different bioinformatic modules as part of an overarching clinical, diagnostic, or research application. For example, operations associated with those other modules could potentially be requested from within GUI objects managed by the annotation module.

4.2 Image biomarkers (and others) for cardiac and oncology diagnostics

From one perspective, there is nothing special about image-annotations such that GUI objects within that domain would be a distinguished starting point for bioinformatic operations in general; in principle any window in a biomedical application could potentially be the starting-point for user actions that lead to capabilities realized across two or more modules. On the other hand, bioimaging is an important case study for medical *software,* because in this context the entire data-acquisition chain occurs "*in silico*." Unlike tests based on lab equipment, for example (e.g., colorimetric assays), or clinical evaluations that rely on practitioners' subjective judgments, every step in the image-analysis process is performed via computer code that itself can be evaluated, and all data generated during diagnostic/predictive processes can (at least in principle) be preserved (whereas in a typical blood workup, say, samples are discarded once the experiment is concluded). Moreover, analyses can be exactly repeated by running the same computer code against the same images.

It is also true that because image analyses are already in digital form, there is no extra data-entry step needed, nor ambiguities arising from imposing numerical measures on subjective evaluations (cf. the Medical Research Council scale for muscle-strength we mentioned in Chapter 3). Applying Computer Vision algorithms can yield large quantities of numeric data from image-analysis, and while not all feature-signatures will have obvious biomedical interpretations, the sheer volume of raw data can be beneficial for machine learning, which will sometimes discern statistical correlations that impugn diagnostic or prognostic value to mathematical radiomic entities that may be too subtle for the naked eye. The value of such *a priori* quantification is advanced further, too, by conventions that codify imaging results, such as **RadLex** (for radiology) [41], [61] or, more recently, IBSI, the Image Biomarker Standardization Initiative (centered on radiomics) [93], [94], [3], [37], [27].

These are among the factors that drive the search for reliable image biomarkers, envisioned as essential component in biomedical research/treatment ecosystems. Other analogous factors include how image-based diagnostics can be quicker and cheaper than lab-based alternatives. More broadly, imaging technologies preserve a more holistic data space than (say) biopsies, which by definition excise biologic material from its *in vivo* context. Images preserve a record of spatial relationships (or spatio-temporal relationships, if we consider 4D media, such as flow MRIs), which is lacking from data obtained via gene sequences, blood, or tissue samples. As expressed in an "Assessment of Imaging Informatics for Precision Medicine in Cancer," for instance:

> Educating clinicians on the benefits of imaging methods in clinical practice is key to their adoption [because] in many cases ... genomic analysis alone is sometimes inadequate. Genomic analysis will not

> reveal carcinoma versus benign growth and mutations analyses alone cannot provide a specific diagnosis. [I]n the case of Leiomyoma (benign disease) vs. Leiomyosarcoma (cancer), the genetic mutation is the same, but human cognition and the use of microscopes are required to accurately diagnose cancer versus benign growth ... Spatial phenotypic heterogeneity is not captured by genomic data. There is no way of understanding interactions between the various cell types in a tumor microenvironment (TME). If the cell composition is the same, but the interactions are different, in two different TMEs, genomics cannot tell them apart. Hence, the study of images, and their spatial data, is crucial. ***[10, pages 6–7]***

This assessment also stresses the importance of sharing image-based data and observations in a consistent manner:

> Integrating radiology, pathology, clinical, and -omics data requires that image annotations be stored in a standardized and interoperable manner. ... Frequently, the annotations, created on commercial image viewing workstations, are collected and stored in either proprietary formats or as DICOM presentation state objects, which are like graphical overlay objects. This enables rendering the information visually, but does not support search of, and access to the annotations, nor any computation on them... Consequently, ... there is no interoperability of image annotations across platforms and applications. To realize the potential value of integrative radiology-pathology-omics, it is vital that image annotations be stored in standardized interoperable formats such as the Annotation and Image Markup (AIM) standard or DICOM ... DICOM Working Group 8 is working to harmonize and unify the AIM and DICOM standards and create a DICOM Structured Reporting object to store AIM image annotations [and] provide a standardized interoperable format for image annotations. ***(page 5)***

In short, bioimaging (including radiomics) is not likely to entirely replace diagnostics/prognostics via other means, but image biomarkers may well substitute for other kinds of biomarkers in some contexts (as a more cost-effective option, say), or may serve to reinforce or confirm other analyses. In an overview of "radiogenomics," for example, [55] expresses the value of radiomic markers as follows:

> [Whereas] "Radiomics" refers to the high-throughput extraction of quantitative features from images, i.e., conversion of images to mineable data, and subsequently using these data for decision support, including patient outcome ... "Radiogenomics" or "imaging genomics" refers to the study of the associations between radiomic data (imaging features) and genomic patterns. ... Such imaging data and associated radiomics may serve as a "virtual biopsy," which is non-invasive, includes the entire tumor, and is repeatable ... and may yield a quantitative predictive signature for advancing precision medicine. ***(page 7)***

See also [64], which stresses the importance of quantitative imaging as a "decision support" tool for

> this era of personalized medicine in oncology [where] we have a responsibility to collect as much meaningful information from different modalities as possible, which can help to make better informed decisions. ... Quantitative imaging is able to contribute significantly to decision support for 3 major reasons: 1) virtually every patient with cancer is imaged with CT, MRI, and/or PET; 2) these images are obtained from the entire tumor, along with metastases, and thus can be used to describe and classify heterogeneity; and 3) these images can be obtained routinely longitudinally to monitor responses and to guide specific therapies. ***(page 12)***

For these sorts of reasons, and also because image-annotation marks a good case study for modular design, it is reasonable to consider scenarios where user requests within clinical software originate from a bioimaging context. That is, we will focus on patterns where the specific software-operational sequences under consideration begin with users working within a bioimaging/image-annotation module, and may then proceed to examine other (related) content.

Certainly there are numerous pathways wherein an annotation-related GUI object could be the origin point for actions leading elsewhere. For example, many ground-images presented in the context of bioimage annotations would presumably be associated with an image series taken in a diagnostic context and/or from a spe-

cific patient; as such, consistent with principles of responsive user interface design, users could plausibly be given the opportunity to request more information about the image series, the diagnostic context, or the relevant patient, from annotation-related visual objects (e.g., context menus activated relative to the ground image). This could potentially lead users to any genre of data related to the patient or diagnosis, information that might in turn encompass a wide range of data types, some of which we will review here.

4.2.1 Image registration and radiomics for cardiac care

Our first overview will provide one example of how image analysis generalizes to other bioinformatic domains: we will specifically look at cardiac diagnosis, and in particular image feature-extraction using techniques associated with radiomics, which has been more widely applied to cancer/oncology. The term "radiomics" overlaps with what in more general contexts (not restricted to bioimages) one might call feature-extraction (adopting the "-omics" suffix to suggest parallels with genomics, proteomics, transcriptomics, and so forth [33, page 4]), although *radiomics* specifically tends to be applied in contexts where large feature-vectors are extracted, and then statistically analyzed to find diagnostic correlations. In short, radiomic methods are not contingent on *a priori* anticipations that any given image feature would have an established biomedical interpretation (the way that ground-glass opacity on lung scans, for example, is clearly associated with COVID, via well-understood biologic mechanisms).

In the cardiac case, several studies have appeared in recent years which attempted to disentangle correlations between image features and disease expressions, without those correlations being known ahead of time [7], [60], [8], [39]. Concrete results in the cardiac-radiomics literature have tended to focus on image-textures that indicate cardiac lesions and scarring (associated with greater risk of adverse events such as heart attack and strokes), but researchers have also mined hundreds of image-features for evidence of a select few that seem definitively correlated with heart disease.

In one analysis based on over 5000 UK Biobank patients, for example, [8] point to "gray level heterogeneity" as a feature associated with diabetes and smoking, and possibly other sources of cardiac damage (page 9). "Median" intensity (i.e., overall image-brightness) was also elevated in diabetic patients, from a statistical point of view (page 9). These results suggest that diabetes (and possibly other cardiac risk facts) "leads to a global alteration of the myocardial tissue and thus of the overall myocardial appearance in CMR images" such that bioimages of these tissues register as brighter and less uniform than corresponding images for healthy cohorts (page 8). This is an example of imaging patterns for which we can provide a plausible clinical explanation.

Multiple studies have attempted to identify radiomic features that appear to be diagnostically correlated with cardiac damage along these lines. For example, [5] found two signals that were especially strong indicators of myocardial scarring, one involving "autoregression models" for image textures, and one which was histogram-based (we will consider these metrics in slightly more detail momentarily). The UK Biobank analysis also found strong correlations vis-à-vis certain morphological and "morphometric" (as compared to textural) features; for example, healthy cardiac muscle is apparently correlated with the left ventricle (LV) taking on a visible elliptical shape. Heart *disease*, conversely, is indicated by "spherical disproportion (i.e., the inverse of sphericity) of the myocardium at end-diastole" (page 8), meaning that the LV being more spherical just before heart-contraction is a sign of tissue damage. Indeed, the authors also report that the left ventricle has (according to image-based calculations) relatively less surface

area relative to its volume in the presence of decreased cardiac functioning; this intuitively fits the pattern of sphericity, because spheres minimize surface area for their corresponding volume. As the authors suggest, the biologic mechanisms underlying these morphological observations appear to be related to LV hypertrophy: the ventricle becoming enlarged due to exerting greater effort. In general, [8] endorse combining morphological and textural features to develop hybrid image biomarkers strongly predictive of heart risks.

With respect to textural features, several studies have noted the statistical significance of autoregression (AR) models. In the context of image-segmentation, these models are based on the technique of calculating pixel-intensities as weighted sums of the intensity of neighboring pixels. In effect, rather than defining pixel color as a free combination of red/green/blue scalars (or those of some other color basis), each pixel's color (or often just its intensity) is determined as a linear combination of surrounding pixels. In the same way that two different relatively monochrome regions will tend to have similar color-vectors for pixels inside the region—whereas comparing sample pixels from each region yields vectors which are far apart in color-space—two dissimilar textures within an image will tend to have distinct patterns of linear weights, so that these patterns can partition the image into different regions (analogous to segmentation based on color). These principles give rise to autoregression-based segmentation methods.

Radiomic studies suggest that mathematical descriptions of linear weight-patterns along these lines can also be used—apart from image-segmentation—to quantify characteristics of textures for comparison across images, and therefore potentially as a classificatory tool. We mentioned [5]'s analysis, which found that an AR-based feature was one of two most strongly diagnostically indicative (the other was a first-percentile histogram, effectively delineating the lowest intensity threshold where a region is separated from its background). In [5] three other parameters also showed noteworthy diagnostic correlations, albeit less consistently than the two just mentioned, so that [5] proposes in effect a five-part radiomic signature that can be derived from patients' cine MRI videos. However, as a rule, extraction of radiomic signals does not always point toward scientific *reasons*, or "biologic correlation" [85], for why some imaging patterns and not others tend to track disease conditions (some analyses attempt to bridge this gap; *see* [35], [75], [68, page 6], etc.).

It is not always obvious how to interpret such image-feature diagnostic correlations biologically, because the kind of work we have just summarized tends to seek statistical patterns in large numbers of radiomic signals, without anticipating *a priori* which features are likely to be presented differently in diseased tissues or organs than healthy ones. Some correlations are intuitively plausible. For example, the five most-indicative parameters in [5], just mentioned, include intensity histogram variance, which measures the degree to which brightness levels vary from place to place within a region-of-interest (RoI). The authors also found noticeable signals related to "wavelets," which generally measure the degree of homogeneity or heterogeneity in an RoI, taking into consideration that pattern-(dis)similarity in different directions reinforces homogeneity (or the lack thereof). Intuitively, healthy tissues in many contexts may be more homogeneous than damaged/diseased tissue, or vice-versa. In that sense one might expect that there would be consistent variation in radiomic features derived from pictures of healthy and diseased tissue, respectively, insofar as those features are affected by how textural heterogeneity presents itself visually.

Other discriminative signals have less obvious interpretations. For example, [8]'s findings with respect to autoregression and intensity histograms found particularly strong signals within several specific parameters that are

part of larger parameter groups, e.g., the first percentile was calculated to be statistically more pronounced as a potential biomarker than alternatives, such as the 25th, 50th, 90th, or 99th (it is not clear why a 1st-percentile intensity threshold should be singled out in this context). Also, in the case of autoregression, their analysis implies that patterns in the weight through which pixel-color is influenced by neighboring pixel-color was most pronounced in just one specific direction. It is not clear what geometric phenomenon in cardiac muscle could account for one autoregression direction being more significant than others, without the use of contextualizing techniques such as those introduced by [17].[1] On the face of it, the fact that "first theta" parameters in an AR model would form much stronger biomarkers than features from other theta-directions seems hard to account for.

Other studies which similarly look for diagnostically significant image-features have highlighted different parameters, so there does not yet appear to be a scientific consensus on which sorts of feature-vectors provide bonafide biomarkers for heart disease. One factor which likely contributes to this problem is that image quality and metadata vary from one dataset to another (*see* [53, page 3], for example, for an overview of variance based on equipment and/or image-registration techniques, or [84] for assessment of quality-control methods). According to [39]:

> [R]econstructed images can vary markedly not only in image quality but also in how the heart is presented on an image, including changes in orientation of the heart and differences in the plane of imaging, signal intensity of pixels, and degree of artifacts present on the image. Artifacts or poor-quality imaging can degrade radiomic image analysis. The two image quality factors with the greatest impact on texture analysis (TA)—the most common type of radiomic analysis performed in cardiac MRI—are spatial resolution and signal-to-noise ratio .... and numerous additional ones, including MRI field strength and image slice thickness ... [L]ittle to no study has been done to discern how these factors specifically affect cardiac MRI radiomics. These image acquisition-related factors are a potential source of error in published studies. ***(page 2)***

These limitations, however, do not prevent us from considering how the goals of radiomics affect software design, with respect either to applications used for initiating radiomic analyses or those dedicated to showing their results. Cardiac feature-extraction requires a multi-stage image-processing workflow, which would have to be designed in a standardized (and at least semi-automated) fashion for large-scale deployment of cardiac radiomics. Cardiac imaging is usually carried out by recording full 4D pictures of the heart in action, so a preliminary step is always to select particular 2D frames from a full 4D series.[2] Each 2D image accordingly is associated with data concerning how it is oriented within a 4D context. This orientation is moreover defined in terms of recurring patterns in the hearts' rhythms, as well as the hearts' own 3D morphology and positioning vis-à-vis the human body.

Terms of anatomical orientation, such as "sagittal," "transverse," and "coronal" (corresponding to yz, xy, and xz planes if we consider the x and y axes to extend left/right toward the arms and front/back respectively, and z to measure height off the ground) are relevant for contextualizing bioimages when the anatomic positioning of the organs or tissues visible in bioimages is consequential to their functioning, which of course applies to the heart. The heart's morphological details—divided into left and right halves with distinct shapes, and with the bulk of

[1] Based on [8]'s description of methods it seems most likely that they used AR models built in to MaZda, their analytic software which is also the basis of numerous other cardiac-imaging studies, which establishes "theta" parameters according to directions that remain constant across the RoI.

[2] Of course analyses can be performed in three or four dimensions directly, but much of the existing literature is devoted to feature-extraction in two dimensions only, so that time and plane slices of the full 4D data need to be computed ahead of time.

mass concentrated in the myocardial musculature enclosing the ventricles—are also of significance insofar as these details guide segmentation and registration algorithms applied to cardiac images.

When multiple images are jointly utilized for a diagnostic investigation, image-registration sets up correspondences between points in one image and the "same" points in a second image, the equivalence between them defined in terms of their underlying anatomical locations. Some registration methods in the cardiac context specifically are based on "control points" (which can be manually or algorithmically identified) defined in terms of the relatively fixed morphology of the heart [32, page 120], [67, page 2], [56, page 14], [9, page 8], etc. Image registration is often needed in the context of 2D freeze-frames from a single 4D cardiac image because the heart's motion has the effect of shifting the reference frame oriented to cardiac anatomical features against the axes produced by the imaging device [59, page 1012], [40, page 3].

Also, some analyses of single-patient data employ registration algorithms to coordinate image series acquired via different imaging devices, on the premise that distinct image-acquisition methods are more accurate for specific analytic goals: "As each imaging modality provides unique information and overcomes only certain challenges in cardiac imaging, the physician usually prescribes more than one imaging procedure to gather as much information of the heart's condition before making a treatment decision." [51, page 1]. Registration is also used to normalize images obtained from *different* patients so as to "normalize a population of hearts into a common heart template space." [66, page 31].

Considering these registration and orientation requirements, then, any 2D cardiac image has a fairly detailed anatomic context, which is established, or must in part be calculated, prior to methods such as texture analysis being applied. Each image is oriented from the spatial/geometric point of view against our planar model of the human torso (in the sense that these details define how the 2D region sits within its enclosing 3D space) and against morphological landmarks in cardiac anatomy. Each image may be oriented to other images either in the same series (showing different time or planar slices) or to other heart-images entirely (with the goal of normalizing the image to a generic heart-model or "cardiac atlas" [26], [92], [21], [31], [72], [30]). This registration-related aspect of orientation produces metadata reflecting how control points, deformation, or axis transforms map the current image onto a different target image. Images are also oriented temporally against the structured sequence of cardiac rhythm. The totality of these aspects of orientation can potentially be aggregated into data structures characterizing the image *context* that is logically anterior to image *features* obtained via radiomic analysis.

At the other end of the radiomic pipeline, meanwhile, one obtains feature vectors, such as the five-parameter aggregate identified by [5] as strong diagnostic/classificatory signals. We therefore have two varieties of data structures which need to be joined to particular images: *context* data defining the image's situation in a 4D cardiac representation (which is largely *prior to* analysis logically speaking); and *feature* data derived from morphological and textural analysis (thus largely *after* analysis logically speaking). Connecting these two is the radiomic workflow itself: performing analyses that take the image's context as parameters and compute radiomic features against that background.

Consider these data structures from the point of view of software implementations. The *context* and *feature* data packages bookend the radiomic analysis, and would presumably be relevant to users of the software, whether initiating the analysis in the first place or viewing the results. Of course, the (full) contextual data may itself be available only after a complex process with its own workflow, e.g., image registration

possibly paired with manual "control point" annotations. We'll set this detail aside for discussion and just consider the context data as an overall package: a user reviewing the image context would want to obtain information about how the image is oriented temporally and anatomically vis-à-vis the beating heart, and could benefit from seeing control-points or annotations summarizing image-features employed during the registration process. Orientation relative to the sagittal, transverse, and coronal planes can sometimes be visualized via a 3D diagram of a schematic torso; the **3DimViewer** application, for example, which constructs 3D models from 2D image-series, uses a three-frame viewport rendering an image for each of the three anatomical planes and using a torso-figure to track the position of the planes relative to one another as users scroll on those frames (*see* [16, page 87], for example). These orientations may also be presented numerically.

From a modular point of view, a reasonable software design might stipulate that *context* data is presented in a secondary window that could have some multi-media content, e.g., a torso-diagram showing planar orientation and perhaps a wave-illustration for temporal anchoring in the heart rhythm, as well as rendering of numeric values for these and other contextualization parameters in key-value form. If context-data involves image annotations (such as control points), these could be shown on the primary image-view, but with the user having the option of hiding those annotations (perhaps the context-data annotations could be overlaid semi-transparently when the context-data window is visible).

Similar principles could apply to *feature* data. A secondary "feature" window might display key-value data for radiomic parameters while also showing radiomic features in visual form when appropriate, e.g., via image-intensity histograms for every region-of-interest. Moreover, annotations on the main image (such as overlays demarcating texturally segmented RoIs) could be switched "on" or "off" (and perhaps rendered semi-transparent when the feature-data window is visible). Context menus and drag-and-drop handlers (where warranted) could interconnect all three of these relevant windows (main image, context data, and feature data).

One goal when designing an image-annotation module would be to furnish a common pattern for how the module's windows are organized, and its functionality accessed, which could be reused by multiple applications. When the same module is found in multiple places, those use-points gain the benefit of sharing common interaction-patterns which may be helpful to users (easing transition between applications, for instance), particularly if the module is well-designed and user-friendly. Indeed, one reason to describe the *data* that would need to be handled by an image-annotation module is so developers can get a handle on what users will want to see when they interact with windows provided by the module. As an example, we have offered a very summarial sketch of "context" and "feature" data that would tend to accompany cardiac-imaging use-cases.

In our basic outline, a primary image-window would be supplemented with secondary windows rendering *context* and *feature* data when appropriate. This setup could then be extended to other secondary data profiles. When using the module to *initiate* radiomic workflows, one window could provide a visual summary of the workflow, and/or even a text editor, where the user may compose scripts defining the workflow operationally; this window could then be a starting-point for workflow runs. Of course, normally the actual workflow implementation would depend on other modules, so the annotation module would need to orchestrate data-export and cross-module procedure protocols in coordination with other modules (those data export sites and formats would then be modeled jointly within the module's data model and procedural model). Systematic description of GUI requirements helps to translate data and procedu-

ral models into user-friendly modular designs, because (ideally) procedural capabilities are exposed to end-users in a consistent manner, one which presents the user with similar experiences across different applications, and which is optimized for the specific needs of the module's domain. For example, the part of image-annotation data models related to image-registration (exporting data to registration pipelines) should be formalized in conjunction with specifying how registration data should be visualized (e.g., via control points as annotations).

In addition to workflow definitions, secondary data for image-annotation modules might step outside the imaging context entirely. Recent research, for example, has attempted to refine cardiac image-registration methods by consulting simulation and mathematical models of the heart's mechanics. Mathematical descriptions of cardiac rhythms—and of the associated structural changes to the heart's shape during different phases of the heart-beat—can identify constraints which would also be apparent (projected onto two dimensions, if working in a 2D context) in cardiac images. The "unique combination of ... b-splines in the Fourier domain ... (BSF)" introduced in [89], for instance, "aims to improve the tracking accuracy of myocardial motion by an add-on regularization layer of pairwise image registrations. The proposed framework is designed to enforce spatio-temporal smoothness, cyclic-nature of cardiac motion, and temporal consistency" that is "an 'add-on' regularization framework ... usable on any ... registration algorithm" (page 2). Virtual Reality is likewise adopted in [1] "to dynamically interrogate biophysical and biochemical events in the 4-D domain" (page 2) yielding statistical signature of cardiac motion-patterns (page 6).

These are examples of cardiac motion simulations yielding data that can improve the accuracy of cardiac image-analysis by defining mathematical constraints or statistical properties of cardiac motion, and the heart's geometry at different points in the beat-cycle. Other simulations ground mathematical cardiac models at smaller molecular/cellular scales ([77], [20], or [71], for example). As supplemental data complementing (and potentially guiding analysis of) image data, such simulations are analogous to computational tumor models that we present later. In general, an image-annotation module might need to establish a protocol for viewing information about such simulations as a peer data package (comparable to context and feature data) and/or to interface with a simulation-implementation (analogous to serving as an entry-point for a radiomic workflow). We will return to the *oncology* simulation case later in the chapter.

Before bringing the discussion back to oncology, however, note one further detail in the cardiac context: imaging data may sometimes be integrated with more conventional biomarkers drawn from biopsies, tissue samples, or clinical health records. In [43], for example, the use of image processing to evaluate myocardial fibrosis is double-checked against direct examination of heart tissue (sampled from explanted hearts, obtained after heart-transplant surgery). Programs such as Canada's "HELP" (human explanted heart program) [90] and the UK Biobank (referenced above; and *see* [74]) encourage researchers to cross-reference cardiac radiomics with other sorts of biomarkers. In [4], data analyzed from the UK Biobank reveals correlations between genetic factors influencing details of cardiac anatomy, such as left-ventricular traits (page 1326), with image-derived phenotypes (page 1320). Genetic factors, specifically MicroRNAs, were likewise correlated with both image and tissue data in [23, see page 9]. In short, some research projects that include cardiac imaging also require analyzing tissue samples, biopsies, genetic data, and other non-image biomarkers obtained from patients in conjunction with image-acquisition.

From the software-engineering point of view, this external data should be linked to images

when warranted based on how study-designs provide context for image-acquisition, even if managing that data is not the direct responsibility of image-annotation modules. This raises the question of how clinical, genomic, or histological data should be integrated with different kinds of bioinformatic modules: How should this more general data be packaged so that it may be presented to the user in multiple application contexts, since that data will be relevant in multiple contexts? How should modules request and render data that lies outside their scope (e.g., genetic data in an imaging context)? These questions, which we have noted in terms of cardiac imaging, are also quite relevant to oncology.

4.2.2 From image-annotations to image biomarkers

Some use-cases for diagnostic imaging require only relatively low-level image scanning (by a person or computer), such as visually confirming the presence of a tumor or, say, bone fracture, or calculating a tumor's width (or a fracture's degree of displacement). Modern image-processing and computer vision applications, however, allow for much more detailed algorithms to integrate image data within informations systems designed for predictive analytics and precision medicine. Textural analysis of tumors or lesions, for example, can yield fine-grained classifications of different patient's particular cancers, which may partition cancer patients into more rigorous groups as criteria for selecting treatment plans, or predicting patient outcomes in light of different possible therapeutic interventions.

The question for image-annotation is how to describe the relationships between image data proper and the textural patterns or image features which are interpreted through the lens of these fine-grained biomedical details. Annotation in the case of simpler, visually evident image-patterns (such as a tumor visible as a darker region against a light background) need only be visually marked or circled to call attention to the Region of Interest, whose biomedical significance is assumed to be evident to a qualified diagnostician who inspects the image. Biomarkers derived from more complex statistical processing of image-data, however, can only be fully described by representing the mathematical results which result from sophisticated image-processing algorithms. The domain of image *annotation*, then, tends to merge with that of feature vectors and/or image-processing pipelines.

For concrete examples of these issues, consider cases such as tumor-microenvironment (TME) research. Radiomics can be used to decode signals latent in tumor imaging which indirectly describe how tumors are biologically interacting with surrounding tissue, measured in terms of parameters or processes such as hypoxia (a situation where a tumor lacks oxygen and tends to respond by more aggressively expanding into surrounding tissue), angiogensis and vascularization (where tumors try to coopt blood supply by spawning new blood vessels) and heterogeneity (reflecting different genetic or morphological patterns in different parts of a tumor, which can potentially make the tumor more resistant to therapy).

One challenge when using image biomarkers in the context of predictive/precision medicine is that of reducing potentially multivariate feature-vectors into signals of just one or two dimensions, which can facilitate grouping patients into clusters of similar diagnostic profiles. For example, [47] discuss hierarchical image segmentation (with specific applications to diagnosing cervical cancer) where contrasts between each region and its enclosing "parent" region provide additional data points (complementing those derived from regions individually). Some regions are composed of smaller ones that have some level of differentiation, and therefore are relatively heterogeneous, whereas other regions are more homogeneous, because their subre-

gions are similar to one another. Measuring heterogeneity and homogeneity across hierarchy-levels allows algorithms to isolate regions that are large enough to be biologically meaningful (smoothing out over-sensitive segmentations that perceive large numbers of small regions due to image "noise").

In particular, important regions tend to be homogeneous at their level, and so on down the hierarchy, but to be children of noticeably more heterogeneous regions at the next higher level. These considerations give rise to a scalar "homogeneity measure" that can be provided via a single formula. In [47], this measure is paired with a metric of region shape based on the eccentricity of ellipses that best approximate each region. The authors call this measure "circularity," which is larger for shapes similar to a circle and smaller for shapes more like a straight line. The homogeneity and circularity measures are single scalar values applicable to each distinguished region. In [47], RoIs are selected as regions that are large in both homogeneity and circularity, followed by a step where RoIs are further classified (using other statistical parameters) as corresponding to cell nuclei or cytoplasm. In short, homogeneity-plus-circularity forms a compact two-valued signature that serves both as an analytic tool and a summarizing device for cellular-scale image segmentation.

Once nuclei are isolated, cancer cells are indicated by nuclei which are enlarged and have irregular boundaries [79, page 4]. This phenomenon applies to many sorts of cancer, although the diagnostic importance of nuclear morphology is more pronounced in cervical cancer than elsewhere, because cervical cancer is commonly diagnosed via blood samples (rather than via imaging solid tumors, for example). Different algorithms can be employed to quantify nuclei deformity, but the common theme is to quantify the deviation of the nuclear membrane from a smooth curve which encloses a similar region [82, pages 4 and 7].[3] The end result is a single scalar estimate of nuclear morphology, which can be applied to all nuclei identified by the prior segmentation. The presence of measurably irregular nuclei correlates with a likelihood of cancerous or pre-cancerous cells, so that these measurements serve as an image biomarker extracted via this form of computer vision pipeline.

This review presents merely one simplified account of a full analytic pipeline. There are many different segmentation algorithms which can be employed to isolate nuclei and cytoplasm: [42, page 2] in a recent (2021) study cite 15 papers describing image-processing methods specific to cervical cytology, and three others for nuclear segmentation more broadly. Image segmentation, and then classification of nucleus (and cytoplasm) regions are two separate analyses, where different methods for each step can be combined independently.[4] Our main point for the moment is that a key step in these analyses is to convert numeric data, whose significance is confined to the intermediate stages of image-processing, into a small group of numbers that can serve as image biomarkers; ultimately integrated into bioinformatic contexts which combine image biomarkers with other kinds of data (genomic, biochemical, histological, and so forth). The analyses we summarized here yield (first) "homogeneity" and "circularity" metrics for each screened RoI and (second) "irregularity" metrics for regions classified as

[3] Informally speaking, techniques can start with a complex contour—viz., the outer boundary enclosing a region—and simplify it to a smooth curve, measuring how much the original contour changes in the process; or, one can proceed in the opposite direction, starting with the curvature one would expect to find on a smooth contour, and measuring how much the actual boundary deviates from these expectations in the neighborhood of individual points.

[4] We are not aware if the algorithms described in the specific papers we cited to summarize examples of the segmentation process and then the classification process have in fact been used together, but they illustrate the kind of workflow endemic to cytological image analysis.

nuclei. This relatively simple system of three parameters encapsulates image-processing routines that could generate thousands of (intermediate) data points during the course of the pipeline.

Although the goal of image-analysis is usually to reduce complex analytic data into simpler, biologically meaningful metrics, there are many different kinds of derived quantities which can be computed as consequential image-features. In [42] such criteria as contour size, average intensity, "solidity" (defined here as a quotient of contour-area against convex-hull area), and "inertia ratio" (essentially the inverted aspect ratio of an approximating ellipse) are identified as visually distinctive qualities of nuclei (page 3) and used for nucleus classification. In another recent review of extant work, [81] identifies several dozen feature varieties:

> Some authors analyzed four parameters: area, integrated optical density (IOD), eccentricity, and Fourier coefficients. Other authors used 16 features: area of nucleus, area of cytoplasm, nuclear gray level, cytoplasm's gray level, and so forth. Some authors acquired nine parameters: mean intensity, variance, number of concave points, area, area ratio, perimeter, roundness, entropy, and intensity ratio. Finally, some other authors used 27 parameters, which included contrast, energy, correlation, and homogeneity. ... It remains to be studied which parameters are more appropriate for cell classification.

Any of these parameters could potentially be employed as image-features that have some diagnostic/predictive significance, which can result in a given image yielding a diversity of realistic biomarkers, even if most such features only have biological interpretations in specific contexts. Moreover, [81]'s list of parameters are centered largely on those characterizing region *morphology*; different metrics can likewise be obtained for describing image *textures* that are evident inside regions, and which also may have biomedical interpretations (e.g., for tumor-microenvironment investigations as cited above vis-à-vis hypoxia, heterogeneity, angiogenesis, and vascularization). In the context of COVID-19 radiology, for example, [18] describes an algorithm for assessing the probability of SARS-CoV-2 infection from chest CT scans, where hypernodes represent high-dimensional vectors (191 dimensions overall) and hyperedges represent k-nearest-neighbors; here each hypernode represents an entire image, mapped to a 191-dimensional feature-vector.

Some research can potentially close the gap between statistically discerned image-features and biological interpretations/explanations or "biologic correlates" [58, see esp. page 1491*ff*] for their statistical significance by simulating cellular-scale or tissue-scale processes that produce patterns latent in an image. In [29], for example, a theory of "habitat characterization" (page 13) yields a scaffolding for image-feature signatures and (incidentally) an account of biodynamic mechanisms causing radiomic patterns:

> We contend that [image] subregions represent distinct habitats within the tumor, each with a distinct set of environmental selection forces. These observations, along with the recent identification of regional variations in the genetic properties of tumor cells, indicate the need to abandon the conceptual model of cancers as bounded organlike structures. Rather than a single self-organized system, cancers represent a patchwork of habitats, each with a unique set of environmental selection forces and cellular evolution strategies. For example, regions of the tumor that are poorly perfused can be populated by only those cells that are well adapted to low-oxygen, low- glucose, and high-acid environmental conditions. Such adaptive responses to regional heterogeneity result in microenvironmental selection and hence, emergence of genetic variations within tumors. The concept of adaptive response is an important departure from the traditional view that genetic heterogeneity is the product of increased random mutations, which implies that molecular heterogeneity is fundamentally unpredictable and, therefore, chaotic. ***(page 12)***

It is worth noting that variable parameters that govern how image-processing workflows are executed can also serve as biomarkers, or at least as metadata informing how biomarkers

should be interpreted (and as such information that should be included in biomarker packages). In the case of [42], the authors outline seven "tunable parameters" (page 7) which their computer code takes into effect, and which can determine the outcome of the segmentation-and-classification pipeline. Since the image-features that emerge from that workflow are dependent on the initial vector of seven predefined parameters, those parameters are also an intrinsic part of the analytic data, and should be recorded when integrating this data into larger clinical contexts.

Our last few paragraphs, then, have presented concrete examples of how radiomic pipelines convert image data (which has mathematical significance only in the narrow context of image statistics) into meaningful biomarkers. To the degree that image *annotations* are used to represent these biomarkers, the annotations proper must be connected with data structures summarizing extracted image-features. Annotations can interact with image-data on several levels. Since features are often correlated with specific image segments or RoIs, the demarcation of the RoI itself constitutes an annotation that serves as a ground for defining feature data. Also, many RoIs are computed by reference to simpler shapes that are defined in their neighborhood, such as ellipses approximating a region's extent, or polygons forming their convex hull. These simplified shapes can be directly encoded as annotations using conventional geometric descriptions. Finally, feature vectors might be encoded as data structures associated with an annotation in the sense that the annotation describes the region to which the feature-vector applies.

The fact that image-features are usually associated with *regions* (not the entire image) points to a close association between feature-vectors and annotations. On the other hand, the scope of (even a general-purpose) annotation framework does not necessarily extend to thorough descriptions of image-features in general, which may require an entirely different set of mathematical and bioinformatic concepts. As the above discussion has hopefully pointed out, there are literally dozens (if not hundreds) of features that might be quantitatively extracted from an image, and it would be difficult to schematically define all such parameters *a priori*. Moreover, different kinds of image-features, and by extension different image-processing techniques, tend to be associated with different biomedical domains. Segmentation of cellular-scale images for the purpose of nucleus and cytoplasm classification is diagnostically important for some clinical contexts, such as cervical cancer. Other kinds of processing, which may involve very different algorithms, have different clinical rationales. For example, tumor-microenvironment research is focused on images at a different scale (e.g., solid tumors rather than individual cells) and is oriented toward texture analysis more than region-morphology.

We can see the overall field of bioimaging as partitioned into multiple (relatively autonomous) contexts, where the image-acquisition modalities, the applicable forms of computer vision, the interpretations of image-features as biomarkers, and the prototypical processing pipelines can all vary from one context to another. As such, it is premature to schematically outline the domain of biomedical image processing as a whole, rather than modeling such different technological contexts individually. Moreover, these contexts are not static; new imaging technologies as well as new software/computational methods can improve upon existing radiomic techniques, while also consolidating algorithms and workflows that are characteristic of specific diagnostic and investigative domains. For example, enhanced segmentation capabilities emerging over the last decade have apparently consolidated the nuclear-classification pipeline we reviewed in this section as a canonical methodology for cervical cancer detection. Much of the terminology and numerical details intrinsic to this use-case

would be less applicable to, say, tumor microenvironments.

These comments imply that the connections between image-annotations and image-features should be left open-ended, with detailed models of how annotations integrate with biomarkers left to be represented more broadly within computational environments wherein multiple varieties of biomarkers are juxtaposed. The challenge for modular implementations is to provide enough structure for an image-annotation module and radiomics/computer-vision modules to interoperate, but simultaneously to avoid pre-emptively restricting the kinds of data and procedures which each module can take on within its own domain. Finding the proper balance between data and procedural expressiveness/open-endedness, on one hand, and rigorous interoperating protocols, on the other, is a crucial artistry within modular design. We will examine this issue in more detail in later chapters.

4.2.3 Tumor histopathology and simulations

As mentioned above, one line of cardiac research has connected cardiac imaging to simulations and mathematical models of heart-beat rhythms and biomechanics. A similar development may be observed in oncology with respect to computational models of tumor growth and evolution. Tumor imaging and simulations can be mutually reinforcing, in that simulations can help explain how biologic mechanisms within the tumor microenvironment (TME) engender patterns of tissues, vascularization, and tumor growth that can be viewed on radiographic images, whereas image analysis can at the same time double-check simulations' accuracy. Simulating tumor microenvironments (and other pathological or histological processes) helps to expose the causative factors underlying cancer observations, including those warranted by biomarkers. One of the clinical payoffs is more refined precision/personalized medicine: multiscale biological models may clarify the factors driving categorizations such as benign/malignant and between tumors that do or do not respond to radiation therapy, potentially improving automated classifications in ways that are clinically significant—ideally, selecting probabilistically advantageous treatment plans.

When constructing biological models, many tumor simulations combine models of biological processes (at the histological, cellular, and molecular levels, often integrating models at different physical scales) with spatial and geometric simulations of tumor growth and evolution. Such spatial simulations yield geometric prototypes that can be cross-referenced against image biomarkers. That is, accurate tumor simulations may yield predictions about how tumors with specific properties (e.g., specific degrees/regions of hypoxia) would appear observationally in the context of different imaging modalities, such as conventional radiography or newer whole-slide imaging or nano-radiomics: "modeling has provided mechanistic understanding of phenomenological observations based on physical principles and helped establish important quantitative relationships ... spanning several biologically relevant scales in time and space." [19, page 2] To the degree that such models are correct, the predicted observable patterns could then be considered as biomarkers for tumors having their simulated properties. For example, if computational models suggest that tumors in certain circumstances will acquire unevenly distributed but consequential hypoxia (sufficient to diminish the effectiveness of conventional therapies) and predict that tumors in this context will exhibit prototypical textural patterns, then image-processing tools that detect such patterns can be deemed accurate in identifying which tumors have levels and dis-

tributions of hypoxia that should be factored in to treatment plans.[5]

As this example suggests, there is often potential for identifying correlations between simulations and image biomarkers: simulations help us to understand *why* biomarkers actually signal the conditions that they do, which in turn can help us improve biomarkers' accuracy. The synergy between simulations and image biomarkers is accelerated further by the inherently spatial and geometric nature of many algorithms and modeling primitives employed in both areas. For instance, in "unstructured lattice" simulations, individual cells are modeled within a lattice grid and allowed to move independently, subject to system constraints reflecting biological processes (such as cells' access to nutrients, oxygen, and blood flow; *see* [12] for instance) and geometric constraints (such as proper spacing between cells) [86]. Simulations in [78] present an example of lattice-based models generalized to three dimensions. A structurally different lattice-based method, one which permits (rather than forbids) multiple cells on one lattice site, is developed in [24], providing a case study in how related simulations may present different modeling parameters, assumptions, and structural frameworks that need to be properly documented when comparing simulation results and conclusions. The structure of the underlying grid, along with the specific evolutionary constraints recognized for a given simulation, provide geometric and data-field primitives that algorithmically generate the overall model. These examples illustrate how biophysical and systems-biological models are often based on shared (or at least analogous/contrastable) geometric primitives that capture how large collections of smaller-scale units (such as cells and proteins) aggregate into structures evincing holistic patterns (such as tissues and ECM) in the presence of local forces and generative rules.[6]

When comparing simulations and radiomics, it is also worth considering how models are codified and documented. Alongside the code that executes computational-biology simulations—or, correlatively, image-processing workflows—bioinformaticians have also specified formats for representing models' parameters, assumptions, and investigative purpose. Popular model-description languages in oncology and immunotherapy include SBML (systems biology markup language), BNGL (the custom BioNetGen language), BioPAX (biological pathway exchange), TumorML, CellML, FieldML, ISML (insilico modeling language), MIRIAM (minimum information required in the annotation of models) and MIASE (minimum information about a simulation experiment). In addition to special description languages, object-oriented methods for simulating tumor histology and microenvironments via general-purpose programming languages (such as C++) are analyzed in [15] (*see also* [46, page 138]). A project such as the Digital Model Repository, which "allows a model to be executed as an independent computer application" and will "in the future ... enable seamless integration of different computational modules based on the use of various accepted ontologies and semantically annotated objects/parameters exchanged between applications" [88, page 9] employs Semantic Web tools for data *representation* but envisions (pages 7–8)

> having the models and data available in standardized formats with clearly stated dependencies [to] facilitate the creation of workflows that can generate model results to compare model predictions with experimental data, all in an automated fashion. This task can be facilitated by collaborating with other groups developing standard formats for model exchange at different biological scales, such as the

[5] *See* [13] for image-processing algorithms targeting irregular hypoxia patterns.

[6] Similar comments could be made for other mathematical tissue-models, e.g., for surgical simulations; a good overview is [91], or [83], [73], [45], etc.

Systems Biology Markup Language (SBML), CellML, BioPAX, and FieldML

which implies a modular design integrating parsers, simulators, and analyzers cross-referenced with empirical data sources, that is, complex software systems that would almost certainly be engineered on an object-oriented foundation. Object-oriented models of cancer-related simulations (tumor formation, growth, morphology, microenvironment, angiogenesis, vasculogenesis, histological properties, and so forth)—and concurrently of radiomic biomarkers (and biomarker-extraction techniques)—can therefore play two complementary roles: to serve as a basis for model descriptions via languages such as SBML, TumorML, etc., and to implement algorithms for computational-biology simulations and/or image processing. In short, expressing modeling constraints and parameters via object-oriented classes allows model description and algorithm implementation to be tied together, which can then streamline the inclusion of bioimaging content since the vast majority of image-processing libraries are based on (object-oriented) C++.

These considerations suggest a programming strategy wherein—staying with systems biology as a case study—the building blocks of systems-biology/immunotherapy-related data sets or research code libraries would be C++ classes, which simultaneously provide model descriptions (that may be formalized via SBML and related languages, potentially generating descriptive markup code automatically via metatype reflection) and, via class methods, provide algorithmic capabilities. These building blocks could then be composed and aggregated into heterogeneous data sets and cross-disciplinary research programs, insofar as most immunotherapy research combines multiple forms of biomarkers, data sources, and/or investigative procedures.

So long as each component part of such hybrid models are rooted in a coding method that integrates model-description and algorithm-implementation, complex hybrid models will have on aggregate a consistent interface for descriptive modeling and for algorithmic logistics (the testing, programming interface, data-acquisition logic, and similar requirements for using implemented algorithms scientifically). If consistently adopted, such object-oriented architecture would yield more consistently designed and reusable Research Objects (arguably more so than current paradigms, where meta-models and operational requirements are expressed and tested via a diverse and disconnected array of incompatible tools and languages).

As a concrete example, the study of tumor hypoxia (decreased oxygen within tumor tissue, a condition which generally makes solid-tumor cancers more dangerous and resistant to conventional therapies) draws together bioimage markers that can estimate hypoxia via image textural analysis, mathematical and cellular/histological simulations of tumor growth to advance our understanding of how hypoxia emerges in heterogeneous and anisotropic tumor microenvironments, genomic, and proteomic data relevant to proteins that influence tumor hypoxia, and empirical clinical or lab data (including tissue samples and disease progression to correlate with bioimages). Most studies of tumor hypoxia combine three or more of these distinct data profiles.

Given that this is the case, it would be helpful to employ a common programming framework to manage all of the data acquisition, modeling, and analysis requirements that are distributed across these distinct domains. For instance, assuming the underlying programming environment is based on C++, all the pertinent data structures integrated in a given tumor-hypoxia research project could be expressed as C++ objects. These data structures might include image annotations identifying regions-of-interest and quantifiable features (vector fields, diffusion measures, etc.) in tumor images; simulation kernels and evolution parameters for tumor growth models that predict how hypoxia manifests in

tumors; clinical records for comparing predictive modeling with actual disease progression; molecular models for proteins influencing tumor hypoxia; and so forth. By structuring all of these research parameters as C++ objects, scientists would then have a centralized paradigm for describing all relevant observational or computational details contributing to theoretical models and research findings, as compared to a diffuse assembly of narrower models expressed in terms of data-acquisition methods, rather than data models themselves (e.g., XML file types, table formats, and so on).

In the case of tumor hypoxia, constructions such as textural image biomarkers and tumorogenesis simulation parameters are theoretical posits that can be concretized in object-oriented programming idioms, providing a reusable and interactive coding platform that is arguably more convenient and pedagogically valuable—more conducive to experimentation—than static figure illustrations or opaque mathematical equations. These points are well illustrated by several existing publications and code repositories that simulate tumor hypoxia via reusable code, built on top of projects such as **PhysiCell**. The same points similarly apply to other TME factors, such as angiogenesis and cellular density. In short, a unified programming platform providing classes supporting different aspects of tumor microenvironment research, from image analysis and historical simulations to empirical clinical records and proteomic or genomic database queries, would serve both to facilitate implementation of research code and to document research methods, which in turn would add rigor to publications and data sets summarizing the research project.

It is self-evident of course that "in silico" simulations require computer code. More to the point, simulations (at least in contexts such as cardiology or oncology) are not performed in a vacuum, but rather connected to clinical or pathological/diagnostic data either to provide initial parameters to a simulation, or to double-check its results (or both). We'll mention several interesting examples: [62] describes the process of constructing a "virtual patient" cohort by mining real-world clinical data and then using this collection of virtual-patient profiles as the basis for a quantitative systems pharmacology (QSP) model simulating a tumor microenvironment via biochemical equations, a simulation that propagates to a model of tumor evolution that can be used to investigate immunotherapy mechanisms. A multi-part research workflow, as in specifically [44], might combine genomic data (specifically, cancer genomic atlas sequences) with cellular data (from the Broad Institute Cancer Cell Line Encyclopedia), with protein abundance metrics derived via mass cytometry, and with simulations of intracellular signaling and of therapy regimens targeting these signaling mechanisms. The *brain* is used as a "control" standing for "normal" tissue in a simulation comparing tumor angiogensis in a hypoxic environment against blood vessel emergence in non-cancerous organs [54, page 4]. An "effort to integrate mathematical models of cancer with real data in an attempt to develop quantitative, predictive models" [70] is combined with multi-scale mathematical modeling (this article is a good reference overview on mathematical simulations of hypoxia, invasiveness, and other cancer details as well as documenting new multiscale techniques; *see also* [38] that applies these techniques in a context that also emulates real-life patient "subpopulations," in the sense of cohorts within a clinically and sociodemographically diverse human community). In [52], hypergraph-analysis methods are applied to several datasets and mathematical models relevant to systems biology, such as genetic regulatory networks, human-disease networks, and protein complexes; for example, "node statistics or motif detection" may be analyzed on hypergraphs representing interactions between sets of related genes and of related diseases (page 5). Statistical analyses reported by [49] introduce a rigorous quantitative definition of angiogenic

"hot spots" (fluctuations in the density of vascular growth at different regions in a tumor) and demonstrate that such fluctuations are likely to manifest underlying biologic processes, rather than result from chance. Next-generation RNA sequencing is combined with an analysis of tumor samples in [25] and [11] to calculate "a gene-specific model by fitting a smoothing spline with four degrees of freedom to transform RNA-seq data ... into 'microarrays-like' data"; from that transformed data the authors investigate tumor-infiltrating immune cells via microarray-based algorithms, research ultimately targeted at leveraging anti-tumor immune reactions to contain cancers and/or amplify the benefits of cancer treatments [25, page 1037]. Empirical models of "hallmarks"—which are patterns in gene-expression explaining genetic contributions to cancer development that have been incorporated into databases for cancer research—are merged into simulations of cancer development and progression in [63]. Finally, [34] employs decentralized object-oriented coding techniques for "a computational multiscale agent-based model capturing spatially explicit dynamics of tumour development in the presence of adaptive immune response" (page 2).

Characteristic of these various multi-methodological studies is the recurrence of biomarker or prognostic factors in different investigative modalities, such as microvessel density (MVD) in [49], which can be expressed in results from histological assays based on immunostaining [49, page 19163], in whole slide imaging applied to tumor cross-sections, and also as a simulated emergent pattern in, e.g., [48]. The former paper moreover presents a method for merging and cross-referencing bioimaging on two different scales (tumor radiography and tumor histopathologic whole slide imaging), allowing MVD image biomarkers to be derived via two different workflows. In [87], a proteomic biomarker (associated with Ki-67, a protein whose levels are correlated with cell division, and in particular with tumors' aggressiveness) and several other immunohistochemical (IHC) indicators (obtained via tissue reception), as well as basic clinical information (e.g., patient age and "Menopause status"), are juxtaposed with ADC-based (Apparent Diffusion Coefficient) image features, seeking a noninvasive tumor-growth prognosis that would statistically mimic the Ki-67 proliferation index in the context of breast cancer. In [28], histopathologic image features are correlated with genomic, transcriptomic and survival data to derive a classification system for cancer types. The common theme of these research projects is that underlying genetic, biochemical, or biomechanical processes can yield multiple biomarkers expressed in different modes of observation (e.g., tissue examination via lab assays versus non-invasive diagnostic imaging).

In the case of Ki-67, levels of the protein in tumor cells signal the rate that these cells are primed to subdivide (and therefore the cancer's invasiveness). As such, Ki-67 serves as a molecular indicator of tumor characteristics that has prognostic significance. An analogous molecular expression in the context of tumor angiogenesis derives from the "ED-B" isoform (variant) of the protein **fibronectin**, which in general is associated with such functions as cell growth and vascular development. The ED-B form is particularly involved in embryonic development [69, page 1], but is less present in adult tissue (in the absence of cancer), so that antibodies against ED-B can serve as a measure of tumor angiogenesis [2, page 6]. Proteins can of course be indirectly signaled by antibodies against them as well as by genetic factors, insofar as many proteins depend on specific genes to produce them. Personalized immunotherapy often depends on evaluating genomic signatures marking the characteristics of patients' cancers at a granular level, allowing targeted interventional therapies (as we discussed summarily in the previous two chapters).

Transcriptomic indicators (i.e., those based on RNA) have also been widely studied for oncol-

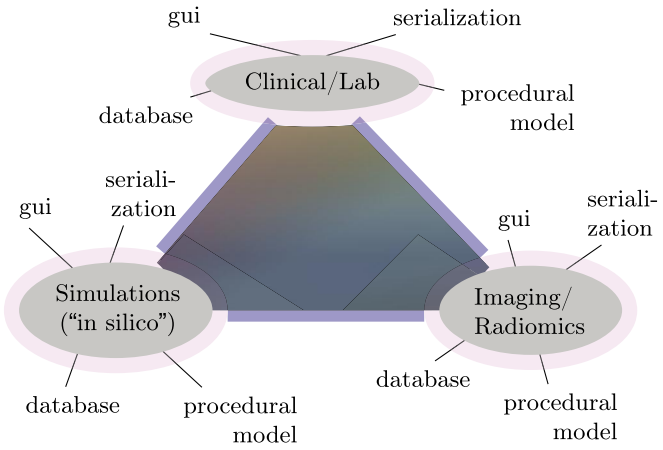


FIGURE 4.1 Triangular relationship between simulations, imaging, and clinical/pathology/diagnostic data.

ogy, yielding for instance biomarkers for hypoxia in the context of cervical cancer, for example [65], as well as hypoxia as a complicating factor in radiation and immunotherapy for many kinds of cancer in general [80, page 6]. In sum, proteins, genes, and antibodies can all serve as indicators allowing diagnosticians/pathologists to infer details about cancer growth and the tumor microenvironment, as well as genomic or molecular factors in the cancer which may have implications for treatment plans and prognoses.

Often similar details may also potentially be gleaned from image-analysis at one or more scales (e.g., radiographic scans of tumors or microscopy images of cancerous tissue), so that image-biomarkers could either reinforce or take the place of lab-based blood work or tissue analysis.[7] Sources of information about cancer properties and TME may therefore involve some combination of (1) non-invasive imaging; (2) tissue/blood lab analysis (proteins, genes, antibodies), which in turn may entertain (2a) cytometry, or lab assays (whose diagnostic mechanisms would comprise, say, color-tests and test-tubes), (2b) image-analysis on tissues/cells (e.g., whole slide imaging), or (2c) genomic/transcriptomic sequencing; and (3) biochemical or biomechanical simulations of tumor growth and evolution. From the computational point of view, these diverse observational modalities give rise to a distinction between software components focused on simulations, imaging, and clinical/lab data management, respectively (see Fig. 4.1). Many research projects combine two or three of these modalities so that this figure (which we will discuss further in later chapters) presents component-types as (vaguely defined) domains that tend to integrate and interoperate in different combinations.

Thus far in this chapter we have discussed the combination and cross-referencing of image biomarkers with other clinical and molecular disease indicators. The various examples and case studies that we have mentioned are of course only a sampling of voluminous research

[7] Of course, image-analysis can replace lab methods in some contexts, but reinforce them otherwise, so that non-invasive imaging can be used for diagnostic/prognostic purposes, but direct lab analysis used as well when patients need to undergo invasive procedures anyhow; and research comparing results from imaging and from more invasive methods can also be done to establish a baseline for how (and how well) imaging results correspond to lab-based results and vice-versa.

literature that could be summarized; a review of the breadth of data-acquisition procedures, observational modalities and research data sources for cardiology or oncology is well beyond the scope of one chapter. Our overview and examples, however, are hopefully sufficient to picture the information landscape so as to present relevant software-design issues. As an organizing device, we have focused on the theme of bioimaging, which translates over to design considerations for software components presenting images (and their annotations) and potentially interfacing with image-processing workflows. Given the correspondences between different forms of biomarkers sketched out via Fig. 4.1, the analogous software-engineering concerns derive from the challenge of integrating components devoted to the three "vertices" of the triangle (imaging, simulations, and clinical/lab data), which we will address in the rest of this chapter.

4.3 Multi-aspect modular design in a heterogeneous data space

In this section we explore the idea of "multi-aspect" modular design, which adds to the underlying principle of modularity the notion of individual modules having multi-faceting software-engineering responsibilities, including (for example) GUI rendering, data persistence/serialization, and runtime reflection or remote procedure capabilities, e.g., exposing a scripting interface for modules' functionality. The above example of a bioimaging/annotation module interfacing with modules in parallel domains (genomics, histology) serves as a plausible illustration of why multiple-aspect design can be implementationally beneficial; in our running example, an imaging module may at some point in time present the user with data which invites follow-up through other modules, but it is outside the scope of the initial (imaging) module to know how such external data should actually be presented.

An image currently being viewed could link to other data structures simply by virtue of its biomedical representatum—cardiac images to echocardiogram readings or heart tissue biopsies, for example; tumor images to genetic tests for proclivities to a cancer variety or tests for the patient's anti-tumor immunological responses. A bioimaging module, e.g., might acquire notifications to the effect that such links are available (as part of the image metadata), but would not know how to display a GUI containing genetic, cytological, or histological data, for example, which is why the actual presentation of that extra-modular data would be deferred to the proper modules. At that point, then, those modules would be presented with contextual information (perhaps an identifier for the current image series, i.e., the on-view image's container, which would likewise indicate the relevant patient) and would have to identify the relevant data within their own scope applicable to that context (they may need to pull relevant info from a data "lake," for example).

Insofar as each module has multi-faceted and mostly self-contained functionality, one module could provide a concise entry-point through which a peer module would be able to derive a more comprehensive package of information to present to the user. Starting from a medical image, say, the user could potentially wish to see a detailed overview of related cytological, histological, or genomic data. Insofar as such data is presented via their own modules, each of these may (from the entry-point provided by the current viewed image's metadata) reconstruct a holistic data package and present this via their own GUI components, so that users would experience the overall progression as switching between (or juxtaposing) image-based views with views characteristic of a separate domain (histopathology, and so forth).

The idea behind multi-aspect modular design is to orchestrate flows of user activity along

these lines in a relatively decentralized manner. Modules that are "multi-aspect" could independently promote a holistic User Experience given a concise entry point—e.g., an image series and patient identifier leading to presentation of genomic or histological data in an integrated GUI component. Modules that were narrower in their range of engineering concerns would require greater central control to permit the "switch" in user focus across domains. If GUI and database access capabilities were separated between two different modules, say, then transitioning from an image-view to (for example) a histopathology view would require a central controller, which *first* loads the relevant histopathology data, and *then* passes this data to a histopathology-GUI component. Having the GUI and database-query functionality merged into a single histopathology module would eliminate most requirements for centralized control, allowing the imaging and histopathology modules to interact on their own.[8]

4.3.1 The overlap between research and clinical data

We have thereby set forward a case for modular design—where modules are interoperable, but relatively self-contained (in the "multi-aspect" sense that they package features related to GUI design, data persistence/serialization, and so forth), and at the same time prioritize minimizing external dependencies and adapting to diverse computational environments, rather than being usable only in a select range of environments, where computational performance can be optimized.[9] Interrelated clusters of code libraries linked to different sub-specialties are provided by some software projects (**BioConductor** is a good example), but these components often do not span multiple aspect and feature-sets with the breadth that we propose via "multi-aspect" design.

As a rule, for example, **BioConductor** packages do not provide built-in database integration or GUI features (except those inherited indirectly from the R environment, which provides a GUI context for evaluating R code in general, but does not natively provide tools for packages to customize the GUI to their needs, without a foreign-language bridge, such as Qt bindings). Reviewing prominent projects for open-source biomedical components, there are libraries that emphasize analytics, UI development (e.g., **OncoJS**), data-set preparation (as with the Research Collaboratory for Structural Bioinformatics Protein Data Bank), APIs for genetic/proteogenomic data access (**EPICO**, GDC, etc.), simulations (**PhysiCell**, **Amber**, **Chaste**, **MMBios**), and so forth, but the components within these projects do not internally address the multiple aspects of (in particular) GUI, database access, and data serialization all together.

Likewise, the three APOLLO (Applied Proteogenomics Organizational Learning and Outcomes) networks—GDC, TCIA, and CPTAC (the Proteomic Data Portal)—all provide some tools to help researchers submit and acquire data sets. Most of these tools, however, are exposed as web services, rather than as code libraries that could be bundled into scientific applications. For example, in the case of CPTAC, the UI code is

[8] Not to imply that central monitoring would be fully ruled out, because one may still intend certain cross-modular interactions to be validated. For example, patient privacy concerns might imply that a technician granted access to one part of a patient's data would not necessarily have authority to view other parts. The point, however, is that central control is not *implementationally* necessary; modules are *able* to orchestrate cross-modular functionality on their own, even if programmers may want to use central monitoring to override potential traffic when appropriate.

[9] Publishers and hospitals, we would argue, have a shared interest in curating software-development tools and partial code libraries that could be leveraged to build modules in both publishing/scientific data-hosting and clinical data management contexts, and in particular modules focused on different biomedical subdisciplines (radiomics/bioimaging, cellular systems, cytology, histology, genomics, and so forth).

designed to provide data visualization capabilities in conjunction with downloaded CPTAC data sets. Specifically, downloads acquired through the Proteomic Data Portal will include HTML files providing interactive plots and other figures summarizing information encoded in the rest of the data set. If CPTAC support is encapsulated in a module embedded in scientific/biomedical applications, similar visualization files may be targeted at the host application, allowing users to visualize the CPTAC data summaries directly, rather than opening a separate web browser to examine the included HTML files.

With respect to CPTAC and TCIA, both of these networks require a multi-step data-acquisition process, which could be streamlined with the help of dedicated software components. The TCIA API is split between two interfaces, and while one of these interfaces may be accessed via a client library provided in Java and Python, these tools have limited value for standalone biomedical applications (*see* [50], for instance). Similarly, the UI tools provided with CPTAC could be generalized to an integrated proteogenomic toolkit combining the CPTAC and GDC UI components. The combined code base would then be available as a suite of GUI classes suitable for embedding in scientific/biomedical applications.

In the context of **MMBioS**, a C++ API—identified as a project-aim in conjunction with curating "a database of molecules, rules, and models that can be used for comparative analysis of existing models and development of new models"[10]—would permit **BioNetGen** "actions" to be called directly rather than through a command line, though perhaps the API could be designed (as is a popular pattern in scientific applications) to support workflows that can be manifest either through command-line actions or directly in code sequences (as well as indirectly via remote procedure calls). In addition to generating command-line invocations, the **BNGAction** Perl module (used to access **BioNetGen** functions for data management and analytics) does rather detailed preparatory checks in some cases, so equivalent functionality would have to be implemented as an API layer. In other words, a C++ API would presumably have several stages, with preliminary logic at one stage yielding data structures to a second API layer that communicates directly with underlying C++ code: C++ data structures would be generated in lieu of command-line invocations.[11] There are numerous options for modeling distributed/asynchronous procedure requests (e.g., whether the response requires a separate parsing/value-extraction step, whether response callbacks carry state, and the specific reactive/function-object mechanisms are used to supply callback procedures); a rigorous API should allow each of these options to be employed when appropriate—itemization of the various cases could be part of (what we call) a "procedural-exposure" model—and should clearly define the protocols and requirements in each case.[12]

Finally, in the case of GDC, a property-graph based data model integrates molecular, clinical, and genomic data according to predefined data types and interconnections. This model is instantiated within computations used by the GDC to validate and harmonize submitted data sets prior to their being made publicly available (data sets may be submitted to GDC, either through the GDC API or via an online portal). However, although the GDC data model is

[10] https://mmbios.pitt.edu/research/technology-research-and-development/network-modeling.

[11] Though at this step it would make sense to allow such data structures to be serialized into workflow descriptions, executed indirectly via command line or RPC and other deferred/distributed methods, etc., as well as being executed directly.

[12] It also appears that much of the data visualization associated with **BioNetGen** runs through a similar Perl/command-line pipeline as BNG Actions, so presumably the API could support visualization, perhaps expanding the range of visualization outputs (maybe full-fledged C++ GUI components instead of text formats such as GML).

clearly documented on the GDC website, the GDC does not provide software tools or code libraries to facilitate the implementation of computer applications that would curate data submitted to and/or acquired from GDC so that researchers could perform their own validation steps in preparation for the GDC analysis.[13] Similarly, the GDC user interface components—which are intended for software providing visualizers for GDC data—are based on **JavaScript**, thereby requiring a **JavaScript** programming environment and/or an HTML rendering engine. In short, the GDC data model, API client, and UI toolkit are each targeted at different programming environments, and the GDC itself does not provide a unified framework that integrates these areas of functionality into a common platform. The GDC's published code and web services document the requirements for applications seeking to interface with GDC data submission, validation, and acquisition protocols, but it is left to third-party software to unify these capabilities into a single platform.

In short, failure to promote software development environments that facilitate the implementation of relatively broad-featured and "multi-aspect" modules contributes to either a pared-down modular design, which (we contend) inhibits data integration, or leans toward the consolidation of software ecosystems, whose building blocks are monolithic applications more than inter-combinable modules, which contributes to ecosystem fragmentation.

There are six or seven facets of data management that come to the fore with database systems in the context of application integration, APIs, scientific data curation, and publication/dataset management—parsers for special-purpose languages; domain-specific query evaluators; custom GUI components; interacting with analytic capabilities, both in-process and out-of-process; metatype systems allowing software components to model scientific processes/phenomena; application integration via type-level serialization and persistence; and dataset/publication integration. This set of concerns reappears in numerous scientific-computing contexts (one can identify similar patterns in bioimage processing, for instance), which suggests that they may serve as a general architectural framework for developing scientific-computing "modules."

Requirements such as GUI design, serialization, and database interop reappear in different contexts in different ways. If we consider research projects that involve some combination of clinical/lab data, bioimaging, and simulations, these concerns will be present in each of these areas (we have tried to connote this visually by inserting concerns diagrammed by a "saltire" as part of the triangular outline in Fig. 4.1; this depiction will be clarified in Chapter 8). Because these different research areas tend to be combined and integrated in different ways—scientists' flexibility to piece together different research projects and paradigms in various combinations is potentially an important source of new discoveries—one could argue that embodying research data and protocols in multi-featured (acting as relatively self-contained mini-applications) but inter-combinable modules is a better software-architectural match to the contemporary scientific landscape than either single monolithic applications or narrower "single-aspect" code libraries that need more centralized oversight to interoperate.

4.3.2 The problem of software ecosystem fragmentation

It is often via an evolutionary and decentralized process that software components, data formats, and analytic methods get consolidated into digital "ecosystems" with their own paradigms and conventions. Digital image-proces-

[13] Although the official GDC API client is a guide for programmers who wish to generate requests against GDC endpoints, it cannot be utilized directly to access GDC data unless applications embed a Python interpreter.

sing is a good example; bioimaging (in particular) tends to gravitate toward certain canonical image-acquisition formats (e.g., DICOM), file types (e.g., TIFF or PNG), and analytic libraries (ITK, OpenCV, etc.). These components fit well-defined roles, allowing multi-component workflows to arise organically even if they are not formally proscribed, rehearsed, or crafted *a priori*. That is to say, software use-cases often embody "informal" workflows, which amount to recurring patterns in how different components' functionality are pieced together to implement what is needed for research projects. Such intuitive patterns are structurally quite similar to formal workflows, even though they are not standardized or officially notated as such. Moreover, software components tend to cluster together based on their coexisting within the scope of informal workflows along these lines.

The converse can also be true: informal usage-patterns might become entrenched into distinct ecosystems even if there is some overlap within their domains and methodology. Perhaps IHC assays evince an example of this phenomenon, being designed to yield visually obvious diagnostic markers, in contrast to sophisticated image-processing methods that detect subtle signals in (say) radiographic images. Because the images generated by laboratory assays such as IHC can be affected by how technicians prepare the imaged tissue samples, these assays are designed to yield signals that are as "unsubtle" as possible; refining the laboratory methods and equipment involves making the experiment more accurate by amplifying the desired observable effect, such as the pattern of staining evident in certain parts of a tissue sample in contrast to the background. For those sorts of reasons, detailed image-analysis is not an intrinsic part of the (informal) workflow usually associated with techniques such as IHC, in contrast to scenarios such as radiographic scans, where researchers have limited control over how tissue images are visualized, *except through* automated image processing.

As a result, the software ecosystem centered on assays such as IHC appears to be somewhat isolated from contexts that rely more on intensive bioimage processing. We have not done a rigorous analysis of usage-patterns (to the degree that such an analysis would even be feasible), so these comments should be considered impressionistic and observational, considering existing literature related to entrenched protocols such as IHC. Yet, as some supporting evidence, attempts such as [57] to refine IHC quantification point to the tendency of IHC to rely on image-viewing software, rather than image-processing libraries, for deriving summarial results. More generally, software ecosystems are more likely to become fragmented when the principal components of those systems are *applications* rather than (in particular) code libraries, or also (say) plugins that can extend applications' functionality. Imaging applications such as **ImageJ** or MaZda provide some image-processing capabilities (e.g., "lasso" tools to grab region contours) and are popular in some scientific contexts (including IHC, judging by literature frequently mentioning **ImageJ** and its peers in discussions of research protocols), but workflows that rely on users manually interacting with applications are less robust and extensible than software ecosystems which can pass data to specialized domain-specific code libraries.

Commercial software products also predominate in many workflows involving special data-acquisition devices, such as biosensors. Flow cytometry, MFP probes, and SPR equipment (just to mention technologies identified in this chapter or Chapter 8) are powered by machines that provide their own software (which has to be custom-implemented, given the unique physical mechanisms and configuration options of the machines involved). Flow cytometry is a good case-in-point: commercial vendors of FCM instruments tend also to provide software applications accessing the data that their equipment generates. Though it would be theoreti-

cally possible for researchers to export FCS (flow cytometry standard) files from the instruments' software and write their own analytic code, most scientists appear to be more comfortable working within the confines of existing FCM applications. This situation is roughly analogous to using image viewers or DICOM consoles for image-evaluation, rather than hand-coded image-processing algorithms. By way of comparison, research projects built around intensive image-processing are more likely to feature a custom code base, with different components sharing data via (at least potentially) automated pipelines.

We alluded several paragraphs ago to bioimage workflows organized around widely used libraries such as DCMTK (for managing DICOM data) and ITK or OpenCV (for image processing). These kinds of components are typically joined together within an overarching code model; for example, a research project may develop a code library that links against both DCMTK and OpenCV, implementing procedures that perform the steps needed to carry data resulting from DCMTK processing over to analytic code featuring OpenCV. Alternatively, DCMTK and OpenCV might form the core of two separate modules, which would interoperate via a command-line interface, but in this case the individual modules would still be designed with the understanding that their respective functionalities need to be synthesized into an overarching workflow. Individual components, that is to say, can endeavor to intrinsically support functionality allowing them to be used in a multi-modular context, by adopting policies/capabilities such as initializing their working environment via a command-line interface and exporting data according to shared protocols.

In effect, each individual component supplies the requisite capabilities allowing different components to be pieced together, and moreover with sufficient preparatory code these workflows can fully or partially automated. In these sorts of contexts an important step in research design is to develop a code base which supports automated workflows along these lines; the research code orchestrating the flow of data between workflow components and the sequence wherein operations exposed by each component are triggered.[14]

Workflows achieved through custom programming as just outlined can be "informal" in the sense that they are not explicitly described (or conceived as "workflows" per se), but instead simply follow common usage-patterns, where various code libraries have evolved to play specific roles (and to expose functionality and data formats conducive to multi-component interoperation). Nevertheless, these workflows acquire a certain rigor because they are made possible by specific kinds of functionality being provided within each component, notably code for importing/exporting data according to specific formats, and one or more "entry points" (or what we will term in Chapter 6 "meta-procedures") which can be initialized with parameters specifying how the components' specific contribution to the larger workflow should proceed. These sorts of workflows can be relatively flexible, with clearly delimited prerequisites for how they may expand in scope — either by existing components recognizing a wider range of data formats, or exposing new functionality, or via separate components encapsulating extended functionality being designed in a manner that permits their integration into the existing workflow patterns.

Fragmentation of software ecosystems into isolated clusters of components typically used together may still be a problem, but at least

[14]It may be imprecise to describe such workflows as "automated" because custom programming is needed to implement the code which acts as a framework for workflows to be executed (we are not referring in this context to workflows visually designed through a workflow-management application rather than by programming them directly). Once the overarching code is implemented, however, each particular iteration of the workflow sequence can typically be performed without human intervention midstream.

there is a technical foundation for considering the proper extent of workflows' scope — one can identify the particular elements within each component which support their interoperation. These elements are the specific sites in the components' code which would be affected if researchers or programmers were to adopt usage patterns that effectively widen the scope of existing (maybe informal) workflows; one could ask, for example, how difficult it would be to implement support for new data formats (in terms of parsers and runtime representations of new kinds of data given components' existing parsing and representational procedures). Likewise one could consider whether it is practical to implement functionality needed to manage new kinds of data (e.g. the structural or mathematical operations endemic to new data profiles which depart to some degree from those the components has previously targeted) given components' existing architecture and capabilities.

Issues of ecosystem fragmentation are more likely to become entrenched in the context of informal workflows which are, so to speak, "application-driven" in the sense that major components are distinct *applications*, often provided by commercial vendors, rather than (one could say) "code-driven" (where components are code libraries). Insofar as informal workflows take the form of common usage-patterns for distinct applications, the logistics of sharing data and orchestrating the proper sequence of operations tends to rely on human users manually interacting with the applications. Compared with code libraries — which are explicitly designed to be integrated into larger programming environments — monolithic applications generally have fewer features enabling functionality available within the application to be exposed for automated workflows. Moreover, full-fledged scientific applications tend to be difficult to extend, even if they are open-source projects with no commercial impediments to customization.[15]

In short, monolithic applications (compared with more open-ended code libraries) tend to be resistant to context-specific modifications which could allow applications to participate in multiple workflow-like environments. One consequence of these limitations is that informal workflows centered on *applications* rather than on *code libraries* tend to be more rigid, foreclosing the possibility of workflows expanding in scope, which in turn drives and reinforces (what we are calling) "fragmentation."

For a concrete example, we will note in Chapter 8 that Flow Cytometry gating and data visualization, and also viewing data generated by image-processing pipelines (such as identified Regions of Interest) have many parallels with image-annotation. These overlaps suggest that a single suite of GUI components could potentially be used to cover both image-annotations and feature-visualization, and moreover extended to other modes of data acquisition such as Flow Cytometry when they engender image-like presentations. One benefit of unifying distinct concerns along these lines is code-reuse. As long as the domains involved are not prohibitively divergent, broadening the scope of existing GUI environments to accommodate a wider range of use-cases can be more efficient than recreating entire GUI tools *ab initio*, even if wider-ranging code bases could become more complex as they support special-purpose data formats, use-cases and functionality.

[15]For example, complex applications are often difficult to compile from source (as compared to installing a prebuilt binary package) which precludes extending the application by modifying the source code directly — this is especially true for software intended to be run on relatively high-powered computers found in research settings, which may have extensive external dependencies that would be impractical to reproduce on more pedestrian hardware. These kinds of scenarios could stymie a graduate student, let's say, trying to work on some specific extension to the application code on a generic laptop computer.

While it is worthwhile to analyze such tradeoffs between code reusability and complexity, for now we simply note that questions about the proper scope for code components tend to dovetail with data-integration concerns. Consider a scenario where image-annotation, image-feature visualization, and Flow Cytometry GUIs are confined to distinct code libraries. One consequence of this separation is that the respective components will likely employ somewhat different representations for annotation (and gating) geometry, image dimensional data, data set provenance, and similar artifacts which structurally overlap across all three domains. Consider the data generated when a user alters the geometry or visual style of an annotation or Flow Cytometry gate, or an image region segmented/demarcated via tunable processing parameters. The domains of image annotations, features, and FCM are arguably similar enough that representations of user actions along these lines will be strongly correlated, and could be expressed via a common description language. Standardizing the respective components' representation of user actions would be beneficial in contexts where data from two or three of these modalities are integrated, so that projects could maintain a holistic record of users' actions during the course of a research cycle.

Common representations of similar kinds of data are more likely when the components that generate such data are implemented as distinct pieces of an overarching environment (e.g., distinct GUI components within a larger GUI toolkit), or at least are self-consciously designed to be interoperable. Fragmentation of software ecosystems can have the effect of obscuring possibilities for the synchronization of data-representations along these lines. More to the point, the presence of multiple data sources which evince similar data profiles, but express information via structurally discordant data models, impedes the process of data integration because such mismatches end up requiring extra "bridge" code. Granted that standardization efforts try to promote interoperability between components engineered by different teams and companies — as we will argue at the end of this chapter, external interop initiatives can have only limited success when they get layered on a code base retroactively, rather than emerging organically from components adapting from the design phase onward to a modular/interoperative environment.

4.4 Data integration via multi-aspect modules

A reasonable overview of oncology or cardiology research — how clinical and diagnostic results are obtained; how scientists explain the biological mechanisms behind cancer or heart disease and extrapolate disease signatures and prognostic indicators from those explanations — suggests networks of interconnected but narrowly focused research programs. Methods and terminology can vary substantially, depending on how researchers target different scales — for example, analyzing precancerous lesions or cardiac tissue versus larger-frame analysis of heart movements or solid-tumor morphology — and also whether the focus is on *in vitro* or *in silico* experimentation on disease processes in controlled environments, or evaluations of real patients (for diagnosis, prognosis, or selecting among treatment options).

Similar dispersion may be found in the software which threads through these research agendas; clusters of similarly focused research work tend to converge on a particular set of software applications, code libraries, or algorithmic conventions. Here we refer to this clustering-effect in terms of "ecosystem fragmentation," or the tendency of research communities to evolve distinctive patterns of software use and computational paradigms, which may have merit in that they encapsulate "best practices" discerned over time, but can also be somewhat inflexible and isolated.

We argue here for a more accommodating methodology which allows modules encapsulating numerous distinct biomedical disciplines to be connected and combined in flexible combinations. We also advocate for modules which are adaptive to different computing environments, without being laden with complex dependencies or locked in to exceptionally high-powered technologies.

To be fair, many research/diagnostic methods in (say) cardiology and immuno-oncology are quiet subtle, relying on precise computational treatments to derive biologically meaningful findings from faint statistical patterns. As such, software which is adequate for these sensitive computational tasks should be fine-tuned for (metaphorically) amplifying faint signals. Under these circumstances, one might reasonably question whether "fragmentation" is a bad thing; perhaps instead this is the only way for researchers to consistently use methods which yield reliable results.

More generally, one can question the extent to which disparate research projects are truly "integrated" in the substantive body of their work mid-stream, as compared to the handful of general principles, clinical indices, diagnostic protocols, and other practical results which hopefully come into focus as research progresses. Obviously, an important research phase is translational — distilling the science into a relatively simple explanatory or evaluative framework that can fit into existing clinical knowledge and methodology. This may take the form of a few prognostic indicators, or a probability distribution estimating the favorability of different personalized treatment plans. Such data points or recommendations would then enter the clinical record and could be a factor in how physicians proceed, with the overall lab or diagnostic process serving as an integral but self-contained interlude in the larger trajectory of patient care. Ultimately, research is most relevant when it yields protocols that slot in to clinical practice in this manner, but the profound details of the research — the complex science behind single biomarkers or indices — can be largely self-contained within the research work or, to the degree that it has practical applications, to the work localized within a given lab or diagnostic center, whereas mostly just summarial findings become integrated into the larger course of treatment.

If complex research data does not in and of itself tend to flow into the clinical mainstream, and if a given body of research has progressed to the point where established protocols exist to yield relatively simple diagnostic findings and recommendations that are clinically relevant, then it is reasonable to ask why any importance should be attached to the insular or conventionalized nature of computational environments that drive research. It is true that many research projects dip into multiple biomedical subfields at once, and therefore make use of computational resources from distinct "ecosystems," but these are often separated as different stages or facets of the overall research. Well-organized research papers usually provide adequate detail describing methods and instrumentation, including description of software protocols (sometimes including published source code) applicable to distinct phases of the research. This means that competent research work is transparent about its methods at each stage, and claims about how the various smaller parts of the research may fit together, to create a larger scientific theory, can be evaluated conceptually.

Much biomedical research is interdisciplinary —whether the juxtaposition of different methods and perspectives exists within a single paper or more within the interplay between multiple research projects, which take the same problem from different angles—but integrating diverse disciplinary perspectives is often only possible by accepting certain empirical or theoretical results localized within one disciplinary area as a starting point for further integration. Establishing the foundation of theories or data involves work narrowed to that specific area

of research; to the degree that scientists feel confident about component results, the goal of cross-disciplinary integration is one of unifying models accepted as provisionally validated by encapsulating their contributions into a few most important details, poised to be conceptually and operationally merged with contributions from other directions. Interdisciplinary collaboration does not necessarily entail low-level engagement of different scientists on the evidential minutiae that need to be curated by individual research programmes to the degree that they can flow into larger multi-disciplinary paradigms.

Such an account of low-level details—"encapsulated" by research programmes that become, in effect, subtheories in a space of integrative scientific practice—would seem to argue for low-level details being the concern only of small groups of researchers (or, upon translation to operational practice, to technicians who have the isolated responsibility of providing their own well-defined step in a clinical pipeline). This is a plausible picture that might accurately describe an ideal of clinical "division of labor," but it is incomplete (we claim) in several contexts, which we will analyze to conclude this chapter. Specifically, we find this an oversimplified account when considering, *first*, large-scale biomedical data management; and, *second*, the publication, dissemination, and reproduction of research work. We will elaborate on these claims over the next several subsections.

4.4.1 Research dissemination and incremental replicability

So, why can't we consider low-level research details as fully encapsulated within narrowly delineated research projects or clinical practice, which would legitimize what we have called "ecosystem fragmentation"? Here we will address two issues.

First, consider the question of reproducing research. Though the term "replication crisis" may be exaggerated, it is true that scientists in numerous fields—especially medicine—have increasingly prioritized structuring research in ways that promote replicability, and that this tendency is driven by scientists' frustration at failures to replicate prior research [76], [6], [36]. If not a "crisis" (a term which might hyperbolically imply that large swaths of medical knowledge could be discredited), then such failures are surely at least a "problem." Assuming that research work is done rigorously, this is not necessarily a problem which researchers can solve individually; the best any individual scientist can do is pursue progress with the most current theories and research tools/equipment possible, and if new science (e.g., new data-acquisition equipment) disconfirms earlier results, the overall trajectory would still be one of science being continuously refined. What researchers *can* do is structure their methodology to prioritize transparency and reduce the difficulty of recreating the research work as much as possible.

Though laudable in theory, replicability can be complex in practice, since quality research by definition will often involve cutting-edge ideas and/or material—the physical accoutrements of empirical research—such that duplicating the scientific environment is impractical for logistical (even if not conceptual) reasons. As we stressed above, a lot of research—especially in the context of bioimaging—has to tease out subtle patterns from complex image and/or statistical data, often depending on specialized image-acquisition tools, and such methods may depend on sophisticated investigative equipment and/or powerful computer systems, which cannot readily be mass-produced. In these situations researchers can still promote replication by transparently describing their techniques and providing future scientists with guidelines on how to reconstruct their work *in those contexts where* requisite hardware and/or software tools are available, but actually following through on such reproductions may involve logistical and financial hurdles. In that sense, even conscien-

tious research may be difficult to reproduce in practice. Since researchers should *not*, in truth, be discouraged from leveraging advanced scientific tools (since these may be the engines driving new discoveries)—however scarce access may be to them among their peers and their field's community writ large—it would be counter-productive to fret over obstacles to replication to such an extent as to obscure the value of original research to begin with.

Nevertheless, science as a whole can still take steps to minimize impediments to replication. The central dynamic of replicability as a *problem* is that sophisticated research may be hard to recreate, because only select groups of scientists have access to the materials that would enable such replication. Here we can set aside other (in theory more corrigible) source of replication problems, such as poor research design or execution *ab initio*, or failure to properly document methods and assumptions. As scientists become more cognizant of replication issues, it is reasonable to hope that these more superficial hurdles could gradually dissipate over time.[16] The more intractable problem is that breakthrough research may be difficult to emulate precisely because research novelty often requires investigative modalities that are not widely available, or on innovative conceptual or mathematical frameworks that will take time to be digested by the larger community.

Against this background, optimal strategies for mitigating replication issues would not necessarily start from the goal of reproducing entire research projects *tout court*. However, replicability is not an all-or-nothing proposition, where scientists need to re-enact every aspect of a research project to certify a replication success. Instead, it is helpful to think of replicability as *incremental*; as a matter of degree. We should be able to reproduce parts of a multi-faceted research agenda, even if it is difficult to redo every piece of the original puzzle. Moreover, prioritizing replicability can also be seen to include facilitating future researchers' ability to identify what would be *involved* in replication to varying levels of detail or thoroughness. The depth and breadth of a replication endeavor should be something that scientists can fine-tune based on available resources and equipment.

Consider a complex, multi-faceted research project, where logistical barriers (financial requirements, equipment availability, and so forth) would make it difficult to reconstruct the project in its entirety. Future scientists can still approach replication in a couple of different ways. First, they can assess the feasibility of reconstructing the whole project, or at least substantial portions thereof: separate and apart from full-fledged replication there is the question of planning and preparing for a replication effort, or estimating what would be required for such an effort to take place. Second, scientists can decide to re-evaluate some portion of a multi-faceted project. They could attempt to confirm the methodology or validate the data involved in some parts of the project, or to recreate all or part of the project to a limited extent (which may involve less comprehensive data sets, less precise equipment, and so forth, perhaps not equaling the standards of the original project, but still contributing some information to an overall assessment of the initial research work).

These possibilities raise several questions *which can be anticipated by the original research framework*: even if this research is carried out using relatively scarce equipment and (e.g., computational) resources, are there paths to reproduce the work (albeit imperfectly) in a less stringent environment? Can some component parts of the research be re-evaluated and (hopefully)

[16]Research trends toward greater quality and precision, and one manifestation of such progress is better research design (enforced by review boards and funding sources, hopefully) and how scientific writing clarifies methods, data sources/availability, or formal protocol descriptions (through projects such as FAIRSharing, "Research Objects," and MIBBI, or minimum information for biological and biomedical investigations), enforced (hopefully) by publishers.

re-confirmed, even if redoing the whole project is impractical? Enabling some level of *incremental* replication can therefore be considered an indicator of quality research design from the outset.

The idea of "incremental" replication has several consequences from a software point of view. Even if a research project uses very large data sets (and this scale is intrinsic to the investigation's merit) there may still be value in approximating the research methods in the context of "smaller" data. Modestly sized data sets could be employed, for example, to estimate or prototype strategies for reproducing the research as a whole. Small data sets could also double-check the accuracy or programming logic of algorithms and/or implementations featured in the original research. Likewise, scientists might at least examine the feasibility of reconstructing prior research on less advanced but more widely available equipment. Would confirmation of the original work (or, for that matter, contra-indicative results) reinforce (or, respectively, challenge) the original work, or is the original methodology tightly bound to the sophistication of its specific materiel?

These issues point to the domain of replicability being more general than just the full-scale reenactment of prior research work. The larger scope of replication bleeds into metatheoretic framing and pedagogical dissemination of a research programme. Consider scientists evaluating what would be entailed in attempt a relatively broad restaging of some complex research. Should we characterize such pre-replication study as a pedagogical happenstance (the scientists trying to arrive at a detailed familiarity with the prior work so as to estimate the scope of a potential replication) or as a conceptual analysis of the original work? The line between pedagogy and meta-methodology may be hard to define in practice.

In short, the community which might be involved in the full scope of replication could be much larger than just those who specifically take on the role of actually carrying out large-scale reproductions. Aside from explicit and relatively full-fledged replication efforts, we have to consider planning *for* replication, assessing how replication projects may be carried out, replications of smaller parts of multi-faceted work, and so forth. Organizing a research programme so as to facilitate subsequent follow-up, then, is not only a matter of streamlining the process of recreating the original working environment in its totality. It is also a matter of enabling scientists to reproduce part of the research setting, or to recreate the environment in a partial or limited manner, so as to *prepare* a more comprehensive reenactment or to replicate just one *part* of the prior work.

Projected onto the specific domain of software development, these concepts imply that the software ecosystem through which new research is disseminated and assessed would, in concord with broad-based replication paradigms, be considered more widely than just in terms of the logistics of full-scale research re-enactments. Replication involves more than just software, of course. It may require the correct genre of equipment, procurement of tissue or other biologic samples (for *in vitro* or *in vivo* studies), access to data for re-use (or functionally analogous data), and so on. Nonetheless, software in particular is well poised to serve as a case study for (what we have termed) "incremental" replication. The software ecosystem through which research may be *incrementally* reproduced or re-evaluated need not duplicate the software through which the original work was carried out (or even the computational resources which would be needed to fully recreate the original context).

Consider scientists investigating what would be *required* for replication, performing a kind of pre-replication prototype of the original project. They would not necessarily need to work on data with the same scale, or computational resources with the same power, as necessary to rederive the original findings. The goal of such

preliminary replication is not to actually recreate the original work, but rather to prototype the environment within which that work *could* be reproduced. In addition, such "pre-replication" should be deemed an intrinsic aspect of replication in general. The consequence of this idea for presentations of scientific findings is that respecting replicability involves more than transparently reporting on methods and protocols *for the benefit of* scientists who might engage in replication full-court. Replicability also entails anticipating the needs of scientists who may simulate the original research environment in a simplified, more modest, or prototyped fashion *as preparation for* potential replication in the more exacting sense.

The possibility of "incremental" replication is one element which complicates, we believe, the problems associated with (for example) what we have called "ecosystem fragmentation." The arguments for research communities narrowing in on relatively isolated and "insular" clusters of computational paradigms and software applications tend to focus on the needs of reproducing research with a level of detail and sophistication commensurate with the original. Indeed, we concede that it is reasonable for scientists to develop research programmes that narrowly focus on certain specific software and computational resources if these are the only options for subsequent researchers who want to build on or re-examine the original work in an environment on par with that original. However, *incremental* replication complicates this picture. Full-fledged replication does not come out of thin air: it depends on scientists intellectually mastering the original work to a degree necessary to play the part of the original researchers in re-enacting their work; and on some anticipation or prototyping of the environment where the replication will be carried out.

In short, researchers should assume that "replication" does not just mean a small handful of follow-up project structurally analogous to the original work. More broadly, replication also involves partial, simplified, or prototype-like simulations of the original research environment and methods toward pedagogical and preparatory ends. Replicability is facilitated by the relevant research community grasping some of the conceptual and operational details of what full-fledged replication would entail, even if many of those researchers are not in fact in a position to launch a replication project of their own. For this dissemination to work most effectively, the larger community that assesses research should ideally have access to at least a simulacrum of the original scientists' research environment.

Moreover, an ecosystem designed for "pedagogical" recreation of the original environment can be more open-ended and less exacting than the original environment itself, because the purpose of such an incremental ecosystem is to sharpen scientists' understanding of research methods as much as to produce new data. A widely accessible "incremental replication" ecosystem could be built around low-cost materials, open-access (and not prohibitively large) data sets, open-source code, and software components that do not have intractable dependency chains or hardware requirements, and which are not locked in to extra-ordinary computing frameworks (e.g., some version of the research protocol should be enactable on ordinary desktop computers).

In short, support for incremental replication along these lines changes the design requirements for software components that are intended to be part of a research ecosystem. The goal of software in this "incremental" context is not to maximize computational performance, or to achieve scientific breakthroughs by leveraging raw computing power. Instead, software in the milieu of incremental replicability serves the primary goal of providing an accessible and educational window onto the protocols and computational patterns intrinsic to a given research programme. Such an ecosystem should seek to disseminate an operational and granular understanding of a particular research project within

the relevant scientific community as a *precursor* to more thorough reproduction efforts. The software which a community uses to initially conceptualize and model original research need not be the same software as that which powers actual full-fledged replications, but we should on principle consider that broader community-scale understanding to be a prerequisite for the planning and prototyping of replication efforts when they do get put into practice—especially when the original research involves rarefied methods or equipment that takes effort to reincarnate in a new setting.

4.4.2 Heterogeneous health data and curation

The issue of "incremental replication," which we just discussed, points to how original research is disseminated in a larger community than just the set of scientists who may in fact be in a position to reproduce prior work with some level of completeness, especially if that work involves materials that are not accessible to many scientists. Even if only a small subset of the relevant scientific community is in position to feasibly contemplate comprehensive research reproduction, the process of preparing for such replication—and for the replication effort itself to pay due dividends in terms of the larger community's appreciating its results vis-à-vis the original work—would seem to depend on the larger community itself, overall, having some operational and granular understanding of the original work. That goal in turn may best be achieved by presenting the community with concrete tools to reenact the original work in a partial and approximative manner, for pedagogical as well as empirical reasons. Software components encapsulating research work, in these kinds of context, may be addressed to a larger group of scientists than just those who intend to reconstruct the original research in a computational environment on par with the original. In the context of incremental replication, software components also or primarily serve goals related to the conceptual and pedagogical dissemination of the original research frameworks and protocols.

Analogous roles might be played by certain software components in the context of multi-domain bioinformatic/clinical data curation. For the sake of discussion, we will consider biomedical data in the context of relatively large-scale and heterogeneous information-spaces, such as a data "lake," where the goal is to deposit as much information as is practically storable, without constraining the data to fit predetermined schema or representations. Hospitals and other medical institutions have increasingly turned to some form of data lake along these lines, although we will speak here in terms of generic paradigms, rather than specific technologies (we are imaging hypothetical information spaces that instantiate the conceptual notion of data lakes or their peers).

Consider the trajectory of a patient's care between hospital admittance and discharge. It is reasonable to assume that the hospital will accumulate a significant corpus of information about that patient, from a mixture of real-time data collected from devices monitoring the patient's condition, to test results and doctor's observations vis-à-vis the patient's evolving condition. Some of this data will be registered on patients' electronic health records, but other information may simple be discarded, or might be siloed into different contexts. For example, intermediate data used to generate lab or diagnostic results may be retained by the clinical entities (inside or outside the hospital) that contribute findings to the official health record, but such more detailed data (presumably more granular, but also less summarial and reusable) might not be computationally accessible within the hospital's digital ecosystem.

This book has focused on bioimaging as a source for case studies exemplifying issues related to biomedical data in general, so, continuing that (expository) device, consider the

specific situation of hospitals outsourcing diagnostic or prognostic investigations to external imaging centers. Data communications between the two entities (the hospital and the imaging center) would presumably be governed by standards such as DICOM, which should clarify how relevant clinical data would be presented to the imaging center and how summarial results would be sent back. In addition to structured diagnostic reports or treatment recommendations, the information sent back to the hospital might include some images, although not necessarily the full image series relevant to that specific patient and/or study (potentially only the most diagnostically pertinent images might be shared) or the full data generated during image-processing (for example, the imaging results could include, so as to filter out midstream calculations, a more compact radiomic "signature" quantitatively merging extracted image-features which prior research suggests are correlated with the patient's specific medical condition). A larger quantity of information may be retained by the imaging center, e.g., on a DICOM database.[17] There would typically, however, be no automated network connection that would allow the hospital receiving the imaging results to access the center's associated PACS archive, should more granular imaging data be required on their end.

The data "lake" paradigm generally takes a more liberal view of sharing and storing data. In this hypothetical imaging context—again, we are speaking in general/conceptual terms, not analyzing specific implementations—we can envision the relevant hospital and imaging center adopting a more data-hungry protocol, wherein a relatively larger volume of image assets and/or metadata is shared between the two entities.[18] The hospital might maintain a system functionally analogous to PACS, which could retain multiple images for each study (and therefore each patient) as well as a reasonably thorough database of radiomic and radiographic image-features extracted from those images.[19] The net result is that within a hospital network itself anyone with proper access rights would be able to pull up patient images, along with annotations and/or image feature-vectors, in conjunction with other branches of patient data (clinical reports as well as diagnostic results in other media, such as blood or tissue samples or genetic sequencing).

There are various scenarios where greater breadth in retaining patient data may be useful. One would be revisiting earlier findings in light of new information: imagine a patient who returns to a hospital some months or years after a prior visit; doctors could find it relevant to look at images taken during that earlier stretch of care. Or, test results from non-imaging modalities might cause doctors to reconsider how the images are to be interpreted. Similar re-evaluation can be warranted if a patient does not respond to intervention in ways which accord with their prognostic cohort, as established by an initial image-based assessment. Also, re-analyzing the original images via different software or different radiomic/radiological methods might yield variant results. The possibility of gleaning new results from re-examining prior data should not be foreclosed due to lack of data

[17] All PACS (picture archiving and communication system) workstations are typically programmed to maintain a database of images viewed through the system, aggregated in conjunction with patient and study metadata.

[18] We write here in terms of *hospitals,* but similar comments would apply to smaller medical institutions, such as "Urgent Care" centers or even private doctors, assuming that the relevant technology can be down-scaled to the computer systems typical of smaller offices; in principle many private systems could be interconnected into a sort of "virtual" Data Lake.

[19] The details of how such data would be communicated alongside the images themselves would have to be established by the protocol connecting the hospital's and imaging center's computer networks, presumably based on image-annotations.

availability. Finally, there are always possibilities of patient cases being material for subsequent research; image data (and other patient records) may be analyzed in light of the eventual patient outcomes. Were a patient to be entered into a clinical trial, image data may become relevant, insofar as such data is pooled from the cohort of trial participants for statistical analysis or data mining.

These comments for bioimaging would also apply to other biomarkers/indicators, such as those derived from blood or tissue samples, genetic tests, functional examinations (e.g., cardiac stress tests or assessments of cognitive functioning), or observational clinical data. Assuming a single software system is available to access the full spectrum of data thereby curated, that one system would accordingly be an entry point to information instantiating a broad range of data profiles. In such an environment users of the associated software would have opportunities to explore the data sets along numerous paths or directions; images of a patient's heart, for example, might lead towards results for patients' exercise tests, blood work, clinical observations (e.g., tracking hypertension levels over time) and genetic data that might have implications for heart disease. Analogously, COVID-19 patients' lung scans could be linked to SARS-CoV-2 antibody tests, assessments of cognitive functioning, contact-tracing data, and so forth. In effect, the software-design issues here are not only those of storing and maintaining large and heterogeneous information spaces, but also structuring the interface for accessing that data in a flexible manner, giving users freedom to traverse the space in multiple directions, and according to multiple criteria.

It would be difficult to implement such a system effectively without a rigorous modular design, because the range of data profiles would exhibit significant heterogeneity. Consider the case of a GUI window showing cardiac or tumor scans and annotations, which is then linked to a separate window showing histopathology results and a further window showing genomic information. The data structures and computational protocols associated with these three domains—radiomics, genomic, and histopathology—are sufficiently different that it would be difficult for a single GUI "template" to effectively handle all three cases. Modular implementations would allow components to be focused on specific areas: modules for image and image-annotation rendering would be separate than those presenting genomic data and from those devoted to histopathology, for example. Such modules might be implemented by different teams, based on rigorous understanding of the science underlying the forms of data they emphasize, rather than trying to fit into a predetermined mold (for data representation, GUI layout, task organization, functional design, and so forth).[20]

Assume, then, that a hospital system employs a *modular* form of "data lake" software system, where the common heterogeneous data source is accessed by collections of discrete modules, each optimized for specific scientific areas: bioimaging, genomics, cytology, histopathology, hematology, neurocognitive informatics [22], epidemiology, and so forth. One issue is then protocols for interoperation between modules; support for flexible usage patterns implies that users should be able to switch between (or visually juxtapose) views provided by different modules. Continuing the above example, cardiac or tumor images might be linked to modules showing genetic data and results on tissue sample-tests, respectively. The imaging module would therefore need to know first which other modules are potentially linked to the current patient data which the user is viewing, and second

[20]This is not to imply that modularity is necessarily embraced within EMR software—for example, commercial clinical data management vendors appear to develop software in a more centralized manner, but such systems are also often criticized by practitioners for being inflexible and difficult to use.

how to describe this current data so that those peer modules would present information that is relevant to that current context; for example, histological analysis linked to the tumor investigated by the current image series.

4.4.3 Modularity and the clinical/research overlap

One goal in this chapter is to present modular design as a solution to problems arising from "software ecosystem fragmentation," the idea being that encapsulating functionality in *modules* (which can be flexibly combined and modified) as opposed to relatively monolithic and isolated *applications* would counter the tendency of usage-patterns consolidating into disconnected paradigms. Issues of research replication add further considerations insofar as clusters of entrenched usage patterns can hinder replicability, even if the impetus toward quality research is precisely the force that carves out ecosystem boundaries in the first place. That is to say, researchers may repeat similar usage patterns because those are paradigms which are optimal to their research work, and would presumably be so for re-enactments of that work in many cases. In short, *full-scale* replication would seem often to entail future scientists converging on the same computational ecosystem (or at least a functionally similar one) as that surrounding the original work.

As we have argued, the problem with this picture is failing to take *incremental* replication into account. Suppose we grant that for some research work a comprehensive reproduction would (to achieve the best results) use the same or similar software as that providing ambient capabilities for the original. We suggested earlier that, such requirements notwithstanding, full-scale replication may only be feasible (and only maximally useful to the relevant scientific community) to the extent in which researchers have operational understanding of what replication entails; can perhaps perform some partial replication in miniature or as a simulacrum of the original; and insofar as prototypes for the replication are discussed and modeled as part of the research process. In short, "incremental" replication can serve as a precursor to full-scale replication, as a tool for building a deeper conceptual and logistical understanding of the applicable research protocols, and as a pedagogical prompt helping the larger community understand the research with greater depth. Replication reinforces the scientific parameters of a research project: instead of a one-off operation, a project reproduced one or more times becomes abstracted from its precise institutional context; it becomes a pattern that can be restaged with some level of variation from place to place. But grasping research projects as a "gestalt" in this sense is easier if scientists in the larger research community engage with the research work at least to some degree in an active, experimental fashion—not just reading an article but carrying out their own mini-replications, for example, or exploring the software which supports that research gestalt.

Even if a certain narrowly circumscribed software ecosystem is necessary for holistic reconstruction of a research project, then, this is only one facet of replication—a further dimension is the broader dissemination of exploratory familiarity with the research methods, a collective intuition of the research challenges from an operational point of view that serves as a precursor to potential replication; and the software appropriate for this pedagogical and exploratory stage may be different from what is prerequisite for the technical management of replications comparable in scope to the original. Against this background, we would argue that similar architectural models appertain to information systems, such as clinical "data lakes." In the same way that the community of scientists who may be engaged with a research programme at some interactive/operational level is broader than just those with the resources to contemplate holistic replications, likewise a

clinical data space spanned by heterogeneous domain-specific modules will be visited by specialists in many areas, and modules should anticipate being used by a diverse array of practitioners, not only those with technical skills finely tuned to the module's core domain.

In the case of radiomics, for example, complex image-processing software and/or code-libraries may be needed to extract feature vectors with sufficient parametric diversity to support radiomic "signatures" and machine learning algorithms endemic to contemporary immuno-oncology or (say) cardiac genomics. These tools, then, for understandable reasons, tend to from a kind of "sub-ecosystem" understood by experts well-versed in the science and mathematics of statistical image-processing. Part of the role of software serving these technical communities is to permit effective data sharing and communication: consider the case of an image series being re-evaluated by a different practitioner, or two different image series (perhaps testing disease/treatment progression) being compared. The interplay between two different practitioners or teams in these contexts is analogous to the relation between an original research group and the consort which replicates their work. These teams may converge on similar software patterns, because they are simply guided by desire for the most accurate results. Carrying the analogy over to an imaginary hospital context, research-replication might be compared to a hospital prescribing diagnostic imaging, and then sending the resulting image-series to a second lab for re-evaluation, and/or evaluating a second study later in the course of care ("replicating" the original analysis, so to speak). Because the two practitioners/teams (or the same practitioners at two different times) are performing two different analyses, there may not be actual data sharing involved (a replication project does not typically reuse the original data, but rather generates a new data set), but the teams may use similar software in each case, software that is functionally targeted toward image-analysis in particular and is disconnected from the hospital's own data space.

Conversely, consider a modular data lake where at least some radiomics-based capabilities and information models are integrated with the hospital's own data management system. In that case, images and radiomic features would be data aggregates accessible within the total package of patient data, and may potentially be consulted by specialists in different medical areas. Instead of radiomic signatures being curated once in a bioimaging laboratory and then only revisited, if at all, by peer practitioners in a similar technical setting, the radiomic features of a bioimaging module could potentially be explored by a broader community of users navigating through the overall clinical information system. This broader "community of users" is therefore analogous to the community of researchers who may be involved with "incremental replication" of a research project, a larger group than the scientists who might engage in a replication study in detail. Consider the case of interacting with a radiomics modules, and then switching to a histopathology module accessing tissue data drawn from the same cancer patient. An analogous transition in the publishing context might be evaluating a data set used for studying radiomic signatures for tumor vascularization, and then switching to a module providing simulations of blood vessel formation in the tumor microenvironment *in silico*.

Whereas a radiomics *application* could serve the needs of isolated laboratories sharing data with referring hospitals in only limited, predefined patterns, a radiomics *module* would be designed for a broader use-base, something that might be embedded in a heterogeneous data-management system and interoperate with other modules. A radiomics *module* therefore could not assume that it resides in a computational environment tailored to computer vision specifically; ideally such modules would be adaptable in the sense that more esoteric dependencies are optional, e.g., that the code is

not tied to exceptionally recent compiler versions or library prerequisites (even if swapping in alternatives yields a degradation in performance). The purpose of a radiomics module would not be instantiating the most powerful radiomic capabilities that science can offer at the moment (as compared to bioimaging software that may be deployed in a diagnostic lab), but rather to expose radiomic capabilities to a larger community, insofar as radiomic data is intrinsically interconnected to information keyed to other biomedical domains (which in turn would occupy other modules). Analogously, too, this situation is comparable to software enabling "incremental replication" having different and (with respect to user base) broader priorities than replication *tout court*.

The analogy between modules targeting a "data lake" and modular design for "incremental replication" has another angle—if we consider *publishers* as akin to *hospitals* (or other institutions curating heterogeneous clinical data). In a decentralized sense, a publisher's digital platform is indeed roughly analogous to a data lake: assets on such a platform include publications themselves, of course, but also bibliometric data, such as influence factor (e.g., references into and out a publication), and, increasingly, digital resources, such as multimedia content, data sets, and computer code. Taken in totality, such assets collectively function as an information space, whose scope, diversity, and architectural challenges are comparable to biomedical records of a large health-care system. Data sets accompanying publications would be analogous in turn to domain-specific biomedical records, which a hospital (say) might store alongside more generic clinical data, e.g., image-processing annotation and feature vectors reported by an external diagnostic-imaging lab.

In some respects this is more than just an analogy; after all, the kinds of data which may be included in a sufficiently broad-based EHR system (radiomic, genomic, histopathological, etc.) are also curated by data-hosting platforms and dataset-archives associated with research publications. Some of these are taken directly from clinical/diagnostic practice—consider projects such as the Genomic Data Commons (GDC), the Oncology Research Information Exchange Network (ORIEN), Human Protein Resource Database (HPRD), The Human Protein Atlas,[21] BioGPS,[22] The Cancer Imaging Archive (TCIA), the Clinical Proteomic Tumor Analysis Consortium (CPTAC), the International Human Epigenome Consortium (IHEC),[23] or the APOLLO network, the GDC, TCIA, and CPTAC, which we considered earlier.[24]

The term "data lake" is encountered more often in the context of clinical records than scientific publishing. However, scientific publishing platforms could generally match the conceptual underpinnings of "data lakes," particularly if we include research data sets (in many cases the organizations hosting scientific data repositories are not the same as publishers hosting collections of books and articles, but there are some companies that play both roles; and, conceptually as well as looking toward the future, we can envision the two technologies merging, so that publishing platforms of the next generation could well feature publications, multimedia, and data sets coexisting, in a cross-referenced and integrated fashion). We can envision modular design playing a role vis-à-vis publication portals analogous to our proposals for bioinformatic data "lakes." In the biomedical contexts, modular design is appropriate because a data lake will contain information with diverse profiles, some involving idiomatic data structures with specialized algorithmic, persistence, GUI, or user-interaction requirements. The same could be said for data sets associated with (and hosted

[21] https://www.proteinatlas.org/about.
[22] http://biogps.org/dataset.
[23] https://epigenomesportal.ca/ihec/index.html.
[24] Not to mention biobanks, which curate not only patient *data*, but patients' actual tissue samples (for research, rather than just therapeutic/diagnostic purposes) or cell lines.

via) a publishing platform. Different modules could be used to render data sets, depending on their underlying scientific field, just as different modules would be selected to display data packages in a clinical system depending on the data's disciplinary provenance (radiomics, histology, etc.).

More to the point; in relatively open-ended contexts, such as data lakes or publishing portals, software components providing such specialized features will often be better designed as *modules* than as *applications*. Whereas monolithic applications may have the requisite power to work with data to optimal degrees (e.g., diagnostic labs re-evaluating findings presented in a clinical data lake, or scientific teams replicating published research projects in detail), many users would have less stringent requirements, because modules can be interconnected in a manner that invites users to navigate between them. Furthermore, modules need to allow this free-form navigation; being able to co-exist and interoperate with other modules in a dynamic fashion can be more important, from the perspective of modular design, than achieving maximal computational performance.

We will argue in later chapters that these ideas have interesting consequences for such issues as data-sharing protocols and information metamodels. One rationale for emphasizing common data *representations* is that information must often be exchanged between monolithic software components engineered in relatively "closed" environments. In that context the basic units of interoperability are often common data models, and secondarily common behavioral contracts for working with shared data.

Consider, however, a more open-ended development ecosystem, where autonomous parties may contribute functional pieces to large-scale bioinformatics platforms in a modular fashion. Such modules are not data *standards*, but rather fully implemented components that work on data directly—achieving standardization through shared implementation, rather than simply through normative mandates—and can be inserted into multiple applications. The module in one application responsible for some specific information-domain would be sharing data with *the same* module in a different application, not just a component which adheres to the same behavioral constraints. Simply using the same code in two different applications obviates the need to document how disparate components should be behaviorally aligned.[25] And, insofar as an application may not want to be forced to adopt a single code base to provide some functionality, at least alternative libraries could seek alignment with their peers by emulating those peers at a relatively low-level code/procedural level. Open-source code allows for components to be synchronized by studying each other's implementations, rather than through oblique standard-definitions.

In short, if the basic mechanisms of behavioral alignment are *code reuse* or, to similar effect, "implementational alignment," constructions such as data models and meta-representations can be built around concrete procedural models, rather than abstract logical summaries of desired behavior, which has interesting consequences for theories of information representation (including ones mediated by Conceptual Space paradigms). We will return to this discussion in Chapter 9.

References

[1] Arash Abiri, et al., Simulating developmental cardiac morphology in virtual reality using a deformable image registration approach, Annals of Biomedical Engineering 46 (2018) 2177–2188, https://link.springer.com/article/10.1007/s10439-018-02113-z.

[2] Artor Niccoli Asabella, et al., Multimodality imaging in tumor angiogenesis: present status and perspectives, International Journal of Molecular Sciences 18 (9) (2017), https://www.mdpi.com/1422-0067/18/9/1864.

[3] Saeed Ashrafinia, Quantitative Nuclear Medicine Imaging Using Advanced Image Reconstruction and

[25] All code by definition is behaviorally aligned with itself!

Radiomics, Dissertation, Johns Hopkins University, 2019, https://jscholarship.library.jhu.edu/handle/1774.2/61551.
[4] Nay Aung, et al., Genome-wide analysis of left ventricular image-derived phenotypes identifies fourteen loci associated with cardiac morphogenesis and heart failure development, Circulation 140 (16) (2019) 1318–1330, https://pubmed.ncbi.nlm.nih.gov/31554410.
[5] Bettina Baessler, et al., Subacute and chronic left ventricular myocardial scar: accuracy of texture analysis on nonenhanced cine MR images, Radiology 286 (2018) 108–112, https://pubmed.ncbi.nlm.nih.gov/28836886/.
[6] Stefano Canali, Towards a contextual approach to data quality, Data 5 (4) (2020), https://www.mdpi.com/2306-5729/5/4/90.
[7] Irem Cetin, et al., A radiomics approach to computer-aided diagnosis with cardiac cine-MRI, https://arxiv.org/abs/1909.11854.
[8] Irem Cetin, et al., Radiomics signatures of cardiovascular risk factors in cardiac MRI: results from the UK Biobank, Frontiers in Cardiovascular Medicine (2020), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7667130/.
[9] Xiang Chen, et al., Deep learning in medical image registration, Progress in Biomedical Engineering 3 (2021), https://iopscience.iop.org/article/10.1088/2516-1091/abd37c/pdf.
[10] Chakra Chennubhotla, et al., An assessment of imaging informatics for precision medicine in cancer, Yearbook of Medical Informatics 26 (1) (2017) 110–119, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6250996.
[11] Pornpimol Charoentong, et al., Pan-cancer immunogenomic analyses reveal genotype-immunophenotype relationships and predictors of response to checkpoint blockade, Cell Reports 18 (1) (2017), https://pubmed.ncbi.nlm.nih.gov/28052254/.
[12] Yong Chen, et al., Simulation of avascular tumor growth by agent-based game model involving phenotype-phenotype interactions, Scientific Reports 5 (2016), https://www.nature.com/articles/srep17992.
[13] Dmitry Cherezov, et al., Revealing tumor habitats from texture heterogeneity analysis for classification of lung cancer malignancy and aggressiveness, Scientific Reports 9 (2019), https://www.nature.com/articles/s41598-019-38831-0.
[14] David A. Clunie, DICOM structured reporting, http://www.dclunie.com/pixelmed/DICOMSR.book.pdf.
[15] Anthony J. Connor, et al., Object-oriented paradigms for modelling vascular tumour growth: a case study, in: Fourth International Conference on Advances in System Simulation, Proceedings, 2012, https://people.maths.ox.ac.uk/maini/PKM%20publications/354.pdf.
[16] Jingjing Deng, Adaptive Learning for Segmentation and Detection, Dissertation, University of Swansea, 2017, https://cronfa.swan.ac.uk/Record/cronfa36297.
[17] Shihong Deng, et al., Autoregressive image interpolation via context modeling and multiplanar constraint, in: 2016 Visual Communications and Image Processing (VCIP), Proceedings, 2016, pp. 1–4, https://ieeexplore.ieee.org/document/7805448.
[18] Donglin Di, et al., Hypergraph learning for identification of COVID-19 with CT imaging, Medical Image Analysis 68 (2021), https://pubmed.ncbi.nlm.nih.gov/33285483.
[19] Prashant Dogra, et al., Mathematical modeling in cancer nanomedicine: a review, Biomedical Microdevices 21 (2019), https://link.springer.com/article/10.1007/s10544-019-0380-2.
[20] Nancy K. Drew, et al., Multiscale characterization of engineered cardiac tissue architecture, Journal of Biomedical Engineering 138 (2016), https://pubmed.ncbi.nlm.nih.gov/27617880.
[21] Frances Duane, et al., A cardiac contouring atlas for radiotherapy, Radiotherapy and Oncology 122 (2017) 416–422, https://cdn.mednet.co.il/2019/06/A-cardiac-contouring-atlas-for-radiotherapy.pdf.
[22] Włodzisław Duch, Neurocognitive informatics manifesto, Cognitive Sciences (2009), ePrint Archive, https://arxiv.org/abs/2101.03609.
[23] Iacopo Fabiani, et al., Micro-RNA-21 (biomarker) and global longitudinal strain (functional marker) in detection of myocardial fibrotic burden in severe aortic valve stenosis: a pilot study, Journal of Translational Medicine 14 (2016), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5002330.
[24] Grazziela P. Figueredo, et al., On-lattice agent-based simulation of populations of cells within the open-source chaste framework, Interface Focus 3 (2013), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3638480.
[25] Francesca Finotello, Zlatko Trajanos, Quantifying tumor-infiltrating immune cells from transcriptomics data, Cancer Immunology and Immunotherapy 67 (2018) 1031–1040, https://pubmed.ncbi.nlm.nih.gov/29541787.
[26] Carissa G. Fonseca, et al., The cardiac atlas project — an imaging database for computational modeling and statistical atlases of the heart, Bioinformatics 27 (16) (2011), https://academic.oup.com/bioinformatics/article/27/16/2288/253995.
[27] Reza Forghani, et al., Radiomics and artificial intelligence for biomarker and prediction model development in oncology, Computational and Structural Biotechnology Journal 17 (2019) 995–1008, https://www.sciencedirect.com/science/article/pii/S2001037019301382.

[28] Yu Fu, et al., Pan-cancer computational histopathology reveals mutations, tumor composition and prognosis, Nature Cancer 1 (2020) 800–810, https://www.nature.com/articles/s43018-020-0085-8.
[29] Robert A. Gatenby, et al., Quantitative imaging in cancer evolution and ecology, Radiology 269 (1) (2013) 8–15, https://pubmed.ncbi.nlm.nih.gov/24062559.
[30] Tarun Kanti Ghosh, et al., Multi-class probabilistic atlas-based whole heart segmentation method in cardiac CT and MRI, IEEE Access (2021), https://arxiv.org/pdf/2102.01822.pdf.
[31] Kathleen Gilbert, et al., Independent left ventricular morphometric atlases show consistent relationships with cardiovascular risk factors: a UK Biobank study, Scientific Reports (2019), https://eprints.whiterose.ac.uk/157310/1/s41598-018-37916-6.pdf.
[32] Xavier Gilbert Serra, Anomaly Detection in Noisy Images, Dissertation, University of Maryland, 2015, https://drum.lib.umd.edu/handle/1903/18119.
[33] Robert J. Gillies, et al., Radiomics: images are more than pictures, they are data, Radiology 278 (2) (2016), https://pubmed.ncbi.nlm.nih.gov/26579733/.
[34] Chang Gong, et al., A computational multiscale agent-based model for simulating spatio-temporal tumour immune response to PD1 and PDL1 inhibition, Journal of the Royal Society Interface 14 (2017), https://pubmed.ncbi.nlm.nih.gov/28931635.
[35] Patrick Grossmann, et al., Defining the biological basis of radiomic phenotypes in lung cancer, eLife 6 (2017), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5590809/.
[36] Stephan Güttinger, The limits of replicability, European Journal for Philosophy of Science 10 (2) (2020), https://core.ac.uk/download/pdf/237399625.pdf.
[37] Akifumi Hagiwara, et al., Variability and standardization of quantitative imaging: monoparametric to multiparametric quantification, radiomics, and artificial intelligence, Investigative Radiology 25 (9) (2020), https://journals.lww.com/investigativeradiology/Fulltext/2020/09000/Variability_and_Standardization_of_Quantitative.11.aspx.
[38] Sara Hamis, et al., Blackboard to bedside: a mathematical modeling bottom-up approach toward personalized cancer treatments, JCO Clinical Cancer Informatics 3 (2019), https://ascopubs.org/doi/10.1200/CCI.18.00068.
[39] Cameron Hassani, et al., Myocardial radiomics in cardiac MRI, American Journal of Roentgenology 214 (3) (2020), https://www.ajronline.org/doi/pdfplus/10.2214/AJR.19.21986.
[40] Alessa Hering, et al., Enhancing label-driven deep deformable image registration with local distance metrics for state-of-the-art cardiac motion tracking, https://arxiv.org/pdf/1812.01859.pdf.
[41] Yi Hong, et al., Application of standardized biomedical terminologies in radiology reporting templates, Information Services & Use 33 (2013) 309–323, https://content.iospress.com/download/information-services-and-use/isu708?id=information-services-and-use/isu708.
[42] Iram Tazim Hoque, et al., A contour property based approach to segment nuclei in cervical cytology images, BMC Medical Imaging 21 (2021), https://bmcmedimaging.biomedcentral.com/track/pdf/10.1186/s12880-020-00533-9.pdf.
[43] Leah M. Iles, et al., Histological validation of cardiac magnetic resonance analysis of regional and diffuse interstitial myocardial fibrosis, European Heart Journal – Cardiovascular Imaging (2015) 14–22, https://academic.oup.com/ehjcimaging/article/16/1/14/2403445.
[44] Mohammad Jafarnejad, et al., Mechanistically detailed systems biology modeling of the HGF/Met pathway in hepatocellular carcinoma, npj Systems Biology and Applications (2016), https://www.nature.com/articles/s41540-019-0107-2.
[45] Xia Jin, et al., Meshless algorithm for soft tissue cutting in surgical simulation, Computer Methods in Biomechanics and Biomedical Engineering 17 (2014), https://pubmed.ncbi.nlm.nih.gov/22974246.
[46] David Johnson, et al., Semantically linking in silico cancer models, Cancer Informatics (2014), https://journals.sagepub.com/doi/10.4137/CIN.S13895.
[47] Aslı Kale, Selim Aksoy, Segmentation of cervical cell images, in: 2010 20th International Conference on Pattern Recognition, Proceedings, 2010, https://ieeexplore.ieee.org/document/5595797.
[48] Yildirim Karslioğlu, et al., Chalkley method in the angiogenesis research and its automation via computer simulation, Pathology – Research and Practice 210 (3) (2014) 161–168, https://pubmed.ncbi.nlm.nih.gov/24359720.
[49] Jakob Nikolas Kather, et al., Continuous representation of tumor microvessel density and detection of angiogenic hotspots in histological whole-slide images, Oncotarget 6 (22) (2015) 19163–19176, https://pubmed.ncbi.nlm.nih.gov/26061817.
[50] Kathiravelu Pradeeban, Ashish Sharma, MEDIator: a data sharing synchronization platform for heterogeneous medical image archives, in: SIGKDD 2015 BigCHat Workshop, 2015, https://zenodo.org/record/844842#.YPoMs1NKiis.
[51] Azira Khalil, et al., An overview on image registration techniques for cardiac diagnosis and treatment, Cardiology Research and Practice (2018), https://www.hindawi.com/journals/crp/2018/1437125/.
[52] Steffen Klamt, et al., Hypergraphs and cellular networks, PLoS Computational Biology (2009), https://

journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000385.
[53] Márton Kolossváry, et al., Cardiac computed tomography radiomics: a comprehensive review on radiomic techniques, Journal of Thorac Imaging 33 (1) (2018) 26–34, https://pubmed.ncbi.nlm.nih.gov/28346329/.
[54] Chung-Wein Lee, Keith M. Stantz, Development of a mathematical model to estimate intra-tumor oxygen concentrations through multi-parametric imaging, BioMedical Engineering OnLine 15 (2014), https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/s12938-016-0235-5.
[55] Hui Li, et al., Quantitative MRI radiomics in the prediction of molecular classifications of breast cancer subtypes in the TCGA/TCIA data set, NPJ Breast Cancer 2016 (2) (2016), https://www.nature.com/articles/npjbcancer201612.pdf.
[56] Lok Wan Lorraine Ma, Mathematical Methods for 2D-3D Cardiac Image Registration, Dissertation, University of Ontario Institute of Technology, 2016, https://ir.library.dc-uoit.ca/xmlui/bitstream/handle/10155/756/Ma_Lok_Wan_Lorraine.pdf.
[57] Robert D. Lovchik, et al., Rapid micro-immunohistochemistry, Microsystems & Nanoengineering 6 (2020), https://www.nature.com/articles/s41378-020-00205-2.pdf.
[58] Meghan G. Lubner, et al., CT texture analysis: definitions, applications, biologic correlates, and challenges, Radiographics 37 (5) (2017), https://pubs.rsna.org/doi/full/10.1148/rg.2017170056.
[59] Timo Mäkelä, et al., A review of cardiac image registration methods, IEEE Transactions on Medical Imaging 21 (9) (2002) 1011–1021, https://pubmed.ncbi.nlm.nih.gov/12564869/.
[60] Carlos Martin-Isla, et al., Image-based cardiac diagnosis with machine learning: a review, Frontiers in Cardiovascular Medicine (2020), https://www.frontiersin.org/articles/10.3389/fcvm.2020.00001/full.
[61] Jose L.V. Mejino Jr., et al., FMA-RadLex: an application ontology of radiological anatomy derived from the foundational model of anatomy reference ontology, AMIA Annual Symposium Proceedings (2008) 465–469, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656009.
[62] Oleg Milberg, et al., A QSP model for predicting clinical responses to monotherapy, combination and sequential therapy following CTLA-4, PD-1, and PD-L1 checkpoint blockade, in: Frontiers in Cardiovascular Medicine, 2020, https://www.frontiersin.org/articles/10.3389/fcvm.2020.00001/full.
[63] Iurii S. Nagornov, Mamoru Kato, tugHall: a simulator of cancer-cell evolution based on the hallmarks of cancer and tumor-related genes, Bioinformatics 36 (11) (2020) 3597–3599, https://pubmed.ncbi.nlm.nih.gov/32170925.
[64] Sandy Napel, et al., Quantitative imaging of cancer in the postgenomic era: radio(geno)mics, deep learning, and habitats, Cancer 124 (2018) 4633–4649, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6482447.
[65] Anja Nilsen, et al., Reference MicroRNAs for RT-qPCR assays in cervical cancer patients and their application to studies of HPV16 and hypoxia biomarkers, Translational Oncology 12 (3) (2019) 576–584, https://pubmed.ncbi.nlm.nih.gov/30660934.
[66] Yangming Ou, Andreas Schuh, DRAMMS software manual, https://www.cbica.upenn.edu/sbia/software/dramms/_downloads/DRAMMS_Software_Manual.pdf.
[67] Yangming Ou, et al., Validation of DRAMMS among 12 popular methods in cross-subject cardiac MRI registration, in: Workshop of Biomedical Image Registration, Proceedings, 2012, pp. 209–219, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5462118/pdf/nihms861246.pdf.
[68] Nikolaos Papanikolaou, et al., How to develop a meaningful radiomic signature for clinical use in oncologic patients, Cancer Imaging 20 (2020), https://cancerimagingjournal.biomedcentral.com/articles/10.1186/s40644-020-00311-4.
[69] Iacopo Petrini, et al., ED-B fibronectin expression is a marker of epithelial-mesenchymal transition in translational oncology, Oncotarget 8 (3) (2017) 4914–4921, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5354880.
[70] Gibin G. Powathil, et al., Systems oncology: towards patient-specific treatment regimes informed by multiscale mathematical modelling, Seminars in Cancer Biology (2015) 13–20, https://pubmed.ncbi.nlm.nih.gov/24607841.
[71] Sergey F. Pravdin, et al., Mathematical model of the anatomy and fibre orientation field of the left ventricle of the heart, BioMedical Engineering OnLine 12 (2013) 13–20, https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/1475-925X-12-54.
[72] Esther Puyol-Antón, et al., A multimodal spatiotemporal cardiac motion atlas from MR and ultrasound data, Medical Image Analysis 40 (2017), https://www.sciencedirect.com/science/article/pii/S1361841517300890.
[73] Kun Qian, et al., Energized soft tissue dissection in surgery simulation, Computer Animation and Virtual Worlds (2016), https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1691.
[74] Raisi-Estabragh Zahra, Steffen E. Petersen, Cardiovascular research highlights from the UK Biobank: opportunities and challenges, Cardiovascular Research 116 (1) (2020) e12–e15, https://academic.oup.com/cardiovascres/article/116/1/e12/5645361.

[75] Stefania Rizzo, et al., Radiomics: the facts and the challenges of image analysis, European Radiology Experimental 2 (2018), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6234198/.

[76] Felipe Romero, Philosophy of science and the replicability crisis, Philosophy Compass 14 (11) (2019), https://onlinelibrary.wiley.com/doi/full/10.1111/phc3.12633.

[77] Daisuke Sato, et al., Formation of spatially discordant alternans due to fluctuations and diffusion of calcium, PLoS ONE (2013), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0085365.

[78] Abbas Shirinifard, et al., 3D multi-cell simulation of tumor growth and angiogenesis, PLoS ONE (2009), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0007190.

[79] Elizabeth R. Smith, et al., New biological research and understanding of Papanicolaou's test, Diagnostic Cytopathology 46 (6) (2018) 507–515, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5949091/.

[80] Brita Singers Sørensen, Michael R. Horsman, Tumor hypoxia: impact on radiation therapy and molecular pathways, Frontiers in Oncology (2020), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0007190.

[81] Jie Su, et al., Automatic detection of cervical cancer cells by a two-level cascade classification system, Analytical Cellular Pathology 2016 (2016), https://www.hindawi.com/journals/acp/2016/9535027.

[82] Jing Rui Tang, et al., Evaluating nuclear membrane irregularity for the classification of cervical squamous epithelial cells, PLoS ONE (2016), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5065206/pdf/pone.0164389.pdf.

[83] Wen Tang, Tao Ruan Wan, Constraint-based soft tissue simulation for virtual surgical training, IEEE Transactions on Biomedical Engineering 61 (11) (2014), https://ieeexplore.ieee.org/document/6820761.

[84] Giacomo Tarroni, et al., Large-scale quality control of cardiac imaging in population studies: application to UK Biobank, Nature Scientific Reports (2020), https://www.nature.com/articles/s41598-020-58212-2.pdf.

[85] Michal R. Tomaszewski, Robert J. Gillies, The biological meaning of radiomic features, Radiology (2021), https://pubs.rsna.org/doi/pdf/10.1148/radiol.2021202553.

[86] Paul Van Liedekerke, et al., Simulating tissue mechanics with agent-based models: concepts, perspectives and some novel results, Computational Particle Mechanics 2 (2015) 401–444, https://link.springer.com/article/10.1007/s40571-015-0082-3.

[87] Maolin Xu, et al., An analysis of Ki-67 expression in stage 1 invasive ductal breast carcinoma using apparent diffusion coefficient histograms, Quantitative Imaging in Medicine and Surgery 11 (4) (2021) 1518–1531, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7930667.

[88] Zhihui Wang, et al., Accelerating cancer systems biology research through semantic web technology 11 (4) (2021) 1518–1531, Wiley Interdisciplinary Review of Systems Biology Medicine 5 (2) (2013) 135–151, https://pubmed.ncbi.nlm.nih.gov/23188758.

[89] Hadi Wiputra, et al., Cardiac motion estimation from medical images: a regularisation framework applied on pairwise image registration displacement fields, Nature Scientific Reports 10 (2020), https://www.nature.com/articles/s41598-020-75525-4.pdf.

[90] Hao Zhang, et al., The human explanted heart program: a translational bridge for cardiovascular medicine, Biochimica et Biophysica Acta (BBA) – Molecular Basis of Disease 1867 (1) (2021), https://www.sciencedirect.com/science/article/abs/pii/S0925443920303434.

[91] Jinao Zhang, et al., Deformable models for surgical simulation: a survey, IEEE Reviews in Biomedical Engineering 11 (2017), https://ieeexplore.ieee.org/document/8107531.

[92] Xingyu Zhang, et al., Atlas-based quantification of cardiac remodeling due to myocardial infarction, PLoS ONE (2014), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0110243.

[93] Alex Zwanenburg, et al., Image biomarker standardization initiative (reference manual), https://arxiv.org/pdf/1612.07003.pdf, 2019.

[94] Alex Zwanenburg, et al., The image biomarker standardization initiative: standardized quantitative radiomics for high-throughput image-based phenotyping, Radiology 295 (2) (2020), https://pubs.rsna.org/doi/10.1148/radiol.2020191145.

This page intentionally left blank

PART II

Type theory, graphs, and conceptual spaces

This page intentionally left blank

CHAPTER

5

Types' internal structure and "non-constructive" ("NC4") type theory

OUTLINE

5.1 Introduction

In the context of text-based serialization languages, such as XML, data structures can be assembled without mandating conformance to predetermined schema. Assuming there is some core set of "basic" types—such as integers, floating-point numbers, and Unicode character strings (to represent names and phrases in natural languages)—data aggregates can be assembled in a "semi-structured" manner, with different types and groupings juxtaposed in no particular order. This indeterminacy is possible because each part of a data structure is embodied via a textual encoding, and text-streams can be packaged in a relatively free-form manner, with no *a priori* length or structure.

Computer software, by contrast, works with binary resources, i.e., with *binary* (rather than textual) encodings of data structures; in the general case we can see binary encodings as strings of 8-bit integers (i.e., strings of bytes, with values in the range 0-255). Binary resources have to be registered in fixed-size, pre-allocated segments of computer memory. Whereas the build-

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00011-8

ing blocks of data structures in a context such as XML are semi-structured data complexes, the analogous foundation for software-centric data models are *typed values*, or binary resources associated with a type $\mathfrak{t}$, which in turn belongs to a *type system* $\mathbb{T}$. Data models in this sense are closely tied to the details of the relevant type systems: in a given $\mathbb{T}$—what constitutes a type in the first place, how types are combined into aggregates, and so forth. In this chapter we will examine the connection between type systems and *hypergraph* meta-models for data representations.

5.1.1 Cocyclic types, precyclic and endocyclic tuples

In a hypergraph-based modeling environment, *hyperedges* may span three or more nodes (ordinary graph edge connect exactly two nodes). For *directed* hypergraphs (DHs), hyperedges have a *head set* and a *tail set*, each a collection of one or more nodes. The term *hypernode* can be used to designate node-sets which are either the head or tail of a directed hyperedge; to avoid confusion, the nodes inside a hypernode can then be called *hyponodes*. Directed hypgraphs that are *labeled* (generalizing ordinary labeled graphs, which are the basis of Semantic Web data, such as RDF) allow information to be associated with connections between hypernodes. Each labeled hyperedge thereby asserts that a certain kind of relationship exists between the entities or sets of entities grouped on either side of the hyperedge (head or tail).[1]

Labeled DHs accordingly have two different formations for aggregating information: first, hyponodes are grouped into hypernodes; and, second, hypernodes are interrelated via labeled connections (hyperedges). This duality allows hypergraphs to combine paradigms associated with ordinary labeled graphs with data tuples or "records" (e.g., Relational Databases). So DHs evince a step toward universal data-representation frameworks, which are structurally rigorous, but not tied down to specific modeling paradigms.

Analysis of hypergraph models can bifurcate into two branches, then, depending on whether we attend to the formation of hypernodes from hypernodes or to the assertion of inter-hypernode connections, via hyperedges. This chapter will focus on the first alternative.

5.1.2 Cocyclic types for hypernodes

We assume we operate in a context where a type system $\mathbb{T}$ is employed in conjunction with hypergraphs so both hypo- and hyper-nodes receive type attributions. We can then consider what sorts of types should be representable in $\mathbb{T}$ to adequately model the spectrum of hypernodes which may appear in a hypergraph. Without undue loss of generality, we can assume that nonidentical hypernodes do not overlap (i.e., no hyponode is covered by more than one hypernode).[2] Any given graph, then, seen as a static data structure, will have some fixed list of hyponodes for each hypernode (since directed hyperedges are ordered, we can assume that there is an ordering on their head and tail sets, and therefore that hyponodes have a fixed order in their covering hypernodes).[3]

In the models we propose, however, we want to focus on "Procedural" Hypergraphs, which are not necessarily static structures. Instead, each hypergraph has certain evolutionary possibilities, i.e., certain regulated operations by which it can be modified, such as (potentially)

[1] The relation is assumed to be intransitive, in the head-to-tail direction, thereby generalizing "Subject/Predicate/Object" triples in RDF.

[2] This restriction — which we call "disjointness" — can actually be weakened somewhat; *see* [3, p. 82] for details.

[3] Assume that hypernode identity is affected by hyponode order; so the same set of hyponodes cannot appear as a head-set or tail-set in two different edgers where their order would be permutated, since that would violate disjointness.

adding a hyponode to a hypernode (if that is compatible with the hypernode's type). We want, then, a more flexible type mechanism whereby hypernodes can cover a varying number of hyponodes. On the other hand, we also want these hyponodes to have types according to some fixed pattern, to preserve a usable type-attribution mechanism for hypernodes. In effect, if we allow hypernodes to cover an unconstrained list of hyponodes with no type restrictions, there ceases to be a structural means of sorting hypernodes by type structure. The problem is then to free up type "tupling" as far as possible, while preserving a strong type system at the hypernode level.

In this context, we propose the concept of "cocyclic" types to convey a pattern among hyponode types that is flexible but still constrained by strong typing. Let $\mathcal{T}$ be any ordered sequence of types in a $\mathbb{T}$. We will say that $\mathcal{T}$ is *cyclic* if the sequence repeats: every n**th** type is the same, for some n. We will call a $\mathcal{T}$ *cocyclic* if it comprises a cyclic sequence preceded by a fixed-width tuple of types. A cocyclic *type* is then a product-type in $\mathbb{T}$ whose instances are hypernodes wherein their contained hyponodes have types which, listed as a sequence of $\mathbb{T}$ types, comprise a cocyclic sequence. The fixed-width tuple at the start of the hyponode-list we will call the *precyclic* part of the hypernode, while the type-tuple that repeats over the rest of the sequence we will call the *endocyclic* part.

This terminology concerning how types in a patterned sequence "cycle" (as a minimal constraint on type-patterns) is idiosyncratic, as are our later terms classifying procedures defined on types (see Section 5.2.1). Our rationale for proposing new language to describe types' internal structure is that mathematical type theory tends to focus more on the interrelationships *between* types than on the *internal organization* of types (and type-instances). This internal organization is more likely to be theoretically significant when we consider how types implemented in a programming language are simulations of real-world objects; that empirical foundation, of course, is largely outside the scope of formal type theory. In practice, a practical type theory for software engineering needs to consider types both as formal and as empirical structures, which calls for a somewhat new vocabulary to articulate types' intermediate status as partly mathematical artifacts and partly software design patterns.

In any case, our definition of cocyclic types can be extended outside the context of hypergraph type-attributions by considering "data fields" or other components of product-types in lieu of hyponodes. Note that any fixed-length product type (whose instances are fixed-length tuples of values, or, in the hypergraph context, hyponodes) is a cocyclic type with no endocyclic part. Likewise, a "list" or "collections" type built on a single $\mathbb{T}$ type—a list, stack, queue, or deque of $\mathbb{t}$s (meaning a list of $\mathbb{t}$s which grows and/or shrinks from one or another end, or both)—is a cocyclic type with no precyclic part (although an implementation might model these instead with precyclic field tracking data such as the current length of the list). An "associative array" (a.k.a. "dictionary") using one type to index a second—where the *keys* to the dictionary come from a $\mathbb{t}_1$ and the values from a $\mathbb{t}_2$—is similarly (representable as) an endocycle alternating between $\mathbb{t}_1$ and $\mathbb{t}_2$.

The purpose of this framework is generality: similar data structures can be emulated in a type system where tuples have to have fixed lengths, or where varying-length tuples have to contain only one single types. In these cases (what we call) "proxies" (hyponodes uniquely designating hypernodes they are not part of) can approximate the layout of cocyclic types, by analogy to programming languages using pointers or nested tuples to represent dynamically sized collections types. However, the cocyclic type paradigm yields less indirection—less gap between the conceptual pattern represented by a data model and its software implementation—without diminishing the model's computational

realizability. Of course, any individual $\mathbb{T}$ type (in the case of a hypernode with one single hyponode) can be seen as a cocyclic type with a length-one precycle.

Against this background, then, we assume that for any $\mathbb{T}$ with fixed-length types we can generalize to a related system, wherein all types are cocyclic. From here on, accordingly, we assume that any $\mathbb{T}$ under discussion is "cocyclic," meaning that each t in $\mathbb{T}$ is cocyclic.

5.1.3 Channelized types and channel algebra

In any reasonably advanced type system, some types in $\mathbb{T}$ are "function-like": they represent computations which, in some sense, take *inputs* of some type or types in $\mathbb{T}$ and produce *outputs*. Programmers sometimes talk of "pure functions" as computations that map inputs to outputs with no side effects. In most programming languages, however, the description of function-like types has to be more complex. In particular, languages can have various sorts of input; in object oriented programming, for example, some (or in some languages all) functions (called "methods") have a special object or "**this**" input which, in various technical ways, is treated differently than the method's other input parameters. Likewise, procedures have modes of "output" other than returning values—they can throw exceptions, modify input values, or have other side-effects in addition to (or in place of) returning values. Procedures implemented in most programming languages, in short, have multiple mechanisms for "communicating with the outside world"—for getting data with which to complete their given task, and for sharing the results of their operations, or otherwise effectuating some change beyond just computing a result. These alternatives generally get some representation in languages' type systems. For example, in C++, a method (which takes an object as a special **this** value) has a different type than a function where that same object would be passed as a normal argument.

We therefore should assume that $\mathbb{T}$ allows us, in principle, to differentiate function-like types on the basis of multiple *kinds* of input and output, and/or side-effects. It is not sufficient to represent $\mathbb{T}$ as permitting, say, given a single input t_1 (or list of input types) and output t_2 (or again a list of output types), the identification of a type $t_1 \rightarrow t_2$ representing functions from t_1 to t_2. Instead, $\mathbb{T}$ has to model multiple input and output modes. This variation does not necessarily yield distinct types; for instance, in C++, whether or not functions throw exceptions is not normally considered part of their type (you can assign the address of a function which **throw**s to a function-pointer whose declared type makes no mention of exceptions). However, differences in input/output options *could* potentially herald differences in type attributions (e.g., a type system *could* stipulate that procedures which do not throw exceptions may never be deemed an instance of the same type as a procedure that *does* **throw**).

We say that a type system is *Channelized* if function-like types in $\mathbb{T}$ can be designated by describing the "channel complex," or groups of channels aggregated together, specifying the kinds of inputs and outputs a given procedure recognizes (along with the types of values passed in to or returned from each procedure). We use the term *carrier* to designate the resource (e.g., a source-code token, or a hypernode in a graph used to model computer code) holding a value in the context of a procedure. Channels are then (potentially empty) sequences of (zero or more) carriers. Carriers are distinct from values (and from types) because they have states separate and apart from the values they hold: when a procedure throws an exception rather than return a value, for instance, the carrier(s) in that procedure's "**return**" channel will hold no value, which on this theory is a valid carrier-state.

Recent work in mathematical linguistics has developed a form of "hypergraph category theory" that can also be applied to programming language type systems (*see* Coecke *et al.* [4], an article we will return to several times in later chapters). Coecke *et al.*'s motivation for adopting hypergraphs is similar to the idea that interprocedural connections represent "information flows." In this framework, computer software can be thought of as an interconnected system whose architecture can be summarized by graphs: with procedures as nodes, an edge exists between procedures when the output of one procedure becomes an input for a second. Moreover, the theory uses nodes *also* to represent information "entering" the system; in effect, when data being presented to a procedure does not arise as the result of some other procedure, but rather as information obtained via some measurement or observation of external states.

Also, nodes may represent side-effects which can be effectuated by a software system.[4] Suppose a procedure concludes by formulating an instruction that a rectangle of a given size and color is to be drawn on a computer screen. The values describing that desired effect are understood to be "outputs" of the procedure, but instead of their being passed to a further procedure, they are somehow translated to tangible effects in the software's external environment.[5]

Taking these two ideas together—*inputs* which are measurements of external states and *outputs* which are effects *on* external states—in the corresponding graph representations, we then have directed edges that have target nodes, but no source nodes (for state-inputs) or which have source nodes, but no target nodes (for effect-outputs). Authors such as Coecke *et al.* then use hypergraphs to model these patterns by leveraging a generalization wherein Directed Hypergraphs' cardinality, for either head or tail, can be any quantity (including zero). That is, hypergraphs in this category-theoretic context allow for hyperedges with no source or no target, as well as hyperedges with multiple sources and/or targets.

The state/effect systems thereby represented (by head- or tail-empty hyperedges) have a corresponding construction, in practical software, via techniques generally described as "reactive," as in "Functional Reactive Programming". A concrete example is the so-called *signal/slot* mechanism used in C++ within the Qt libraries (*see* [5]). A head-empty "state" edge, on this analogy, corresponds to a *signal* which triggers a "slot" procedure; and a tail-empty "effect" edge corresponds to an operation of "emitting" a signal.

There are various options for generalizing graphs to hypergraphs; hypergraph categories are only one example. Also, technical presentations of hypergraphs are not exclusively mathematical: certain software libraries and graph databases also embody formal hypergraph models, with varying features (for instance, some allow hyponodes to also be hypernodes; some support undirected hyperedges; some allow circular hyperedges, wherein the head set is also the tail set). The "Channelized Hypergraph" (CH) model presented here (previously outlined in [3]) is different in some details than hypergraph categories—including by introducing channels as an extra construction on graphs—but arguably the frameworks are sufficiently close that the Channelized Hypergraph constructions (and therefore Channelized Types) are a plausible extension of hypergraph categories from the viewpoint of integration with Conceptual Space Theory.

[4] Hypergraph categories are not specifically about software, but reasoning about software behavior is cited as a possible application and used as a hypothetical case study.

[5] In practice, this would presumably happen by calling some system kernel function, but in the abstract sense we can treat this as an output that is not passed on to another procedure, but instead effectuates the change in some real-world state.

5.1.4 Constructors and carrier states

Coecke *et al.*'s hypergraph category theory reflects how graphs modeling values only as they pass between procedures—outputs becoming inputs—are necessarily incomplete, because values have to originate from *somewhere*. In practice, some data handled by a software component come from external sources, such as files, packets sent over a network, cyberphysical devices, and so forth. Otherwise, often values come from computer code directly: all programming languages have some notion of *source code literals*, which permit values (at least, those of the most basic types) to be initialized from character strings in source code. For example, the literal token "99" becomes the *number* **99**. One question to be addressed for an applied type theory—i.e., to specify the nature of an applied type system $\mathbb{T}$—is how and which types can be constructed from source code literals in this manner.

Representing values passed between procedures can be seen as a "syntactic" gloss on computer code, because programming language grammars are built around how expressions in each language describe the sequence of function-calls specified to enact some algorithm or calculation. But understanding how different typed values interact with function-calls is also a *semantic* matter, characterizing the semantics of individual types. Given a specific instance v of type t, we may analyze, for example, what sort of procedures can produce v? Is v obtained from some other t-value, or can it be initialized via a source-code literal? Is v a "default" value for t which can be created *ab initio*, with no further input? If v is derived by modifying some alternative t-value, can we identify this prior value, thereby "deconstructing" the construction that yielded t?

Suppose, for instance, that t is a list of signed 32-bit integers. The "default value" for such a type is almost always defined as an empty list. New t-values are created by appending a number to the end of the list. Given a non-empty list v, we can always "deconstruct" v by noting that v is derived by adding some number n to a shorter-by-one list v'. Algorithms that depend on examining the whole list—finding its largest element, or counting how many times a given number appears, or how many elements are larger than some target—can be conveniently expressed by examining the list recursively, each time stripping the last number and repeating the test on the shorter-by-one outcome. To count how many numbers are positive, say, check each n at the end of the list, increase the result by one if $n > 0$, and repeat that process with the smaller-by-one list obtained by "deconstructing" the current v into v' and n. This style of recursive algorithm is endemic to functional programming, where it yields procedures that do not need to employ *iterators* which loop over the elements of a data collection. Recursive procedures can eliminate the loop initializations and tests that can make non-recursive, iterator-based code more cluttered and obscure (arguably, difficult to maintain and static-analyze).

However, this recursive style of programming is only possible if specific metadata is embedded with t values, which allows "deconstructing" t instances into construction *patterns* and allows ts to be reused in a recursive fashion (e.g., repeatedly calling a procedure on shorter-by-one lists). This functionality is not available for simpler in-memory representations of data structures, such as "linked lists" (a sequence of pointers to each value paired with pointers to the next value) or C-style zero-terminated arrays. For t to support recursive algorithms in lieu of iterators, it needs to be expressly implemented in anticipation of this pattern. Conversely, types also need extra functionality (e.g., **begin()** and **end()** methods in C++) to support iterators, and extra functionality (like **operator[]** in C++) to support array-based access.

In effect, the semantics of types is much more detailed than simply describing the kind of values t may instantiate. A "list of numbers" may have one abstract profile, but there are a broad range of practical implementations that build,

traverse, and read numbers from the list in different ways. Two types that have mathematically the same "space" of values may be distinct types with very different programming interfaces. Though it is abstractly true that any non-empty list can be "deconstructed" into a single element and a shorter-by-one predecessor with that element removed, a given list implementation may not support procedures to compute that deconstruction in an efficient manner. As a result, mathematical properties of types' sets of possible values have only limited applicability to types' operational semantics.

This point also reinforces the insight that types, in applied type systems (such as programming languages), are not really mathematical entities; they are digital artifacts designed by an implementer to be programmatically employed in specific ways. When analyzing types, we therefore have to explicate the usage patterns that are facilitated by their implementations. Type systems can expedite this process by allowing types to be described in ways that clarify their intended and expected use-cases.

The first step in describing types' usage, moreover, is to account for their constructors, i.e., for the procedures that create new $\mathtt{t}$ values. Constructors are different from other procedures that output $\mathtt{t}$ values, because constructors are internal to how the type is designed; they are in a sense "part of" the type. In most cases, any programmer can write procedures that return instances of $\mathtt{t}$s in $\mathbb{T}$, but most type systems have restrictions on where $\mathtt{t}$ *constructors* are implemented. Constructors are intrinsic to types in that redesigning constructors for $\mathtt{t}$ is understood to modify $\mathtt{t}$'s interface, whereas simply writing a procedure which returns a $\mathtt{t}$ is not normally seen as "changing" $\mathtt{t}$. Moreover, any "external" procedure that returns $\mathtt{t}$ is understood to call a *constructor* for $\mathtt{t}$ to obtain the value to be the external procedure's outcome, or at least to call some other procedure that calls a $\mathtt{t}$-constructor, etc.—whenever a $\mathtt{t}$ is the *outcome* of a procedure, at some point, during some (maybe nested) procedure, there is a call to a $\mathtt{t}$-constructors. Constructors then become landmarks for identifying the properties of $\mathtt{t}$, because all $\mathtt{t}$-values originate from $\mathtt{t}$-constructors at some point.

The discussion in [3] introduces the idea of "co-constructors," which are conceptually similar to constructors, but which wrap the "real" constructors in separate procedures that can present a streamlined type interface. The technical details of co-constructors vs. normal constructors are not pertinent to this chapter, but suffice it to say that a type system may choose to make *co-constructors* the basic building-blocks of a type interface. On this strategy, code which is not part of the "core" implementation of $\mathtt{t}$ would not call $\mathtt{t}$'s constructors directly, but instead would call $\mathtt{t}$ co-constructors.[6]

Co-constructors are similar to what some programmers call "factory methods" or "object factories." Actual constructors have some language-limitations or peculiarities: in C++, for example, you cannot take a pointer to a constructor. On the other hand, insofar as co-constructors are ordinary (non-member) procedures, one can take their address, e.g., for a lookup table mapping strings to co-constructor pointers. That is, co-constructors are more amenable to "reflection," whereby programmers can dynamically invoke a procedure by supplying its name (which in turn is useful for allowing applications to be fine-tuned via scripts, or constructing objects at runtime from a database). As we will argue next chapter, reflection via function-pointers can be a foundation for code-annotation and "documentation by implementation" scenarios. Constructors are also sometimes "default-implemented" by compilers behind the scenes. Co-constructors or "factories,"

[6]The same applies for $\mathbb{T}$ understood not as the system embraced by a *language*, but rather the norms adopted by a library or application: in C++, for example, developers could enforce a framework of co-constructors by strategically excluding (what C++ would call) actual constructors from classes' public interface.

then, are in some contexts more precise representations of types' intended usage patterns than the actual constructors as recognized by compilers.

In particular, implementers of a type `t` may use co-constructors to document and differentiate patterns in how `t` values are created. Constructors for a `t` can be classified into several patterns, such as:

- Default constructors that require no further inputs. Default-constructed values may be deemed conceptually valid instances of their type (e.g., `0` is a valid integer) or may also be "special" values indicating missing data (like null pointers or "`NaN`").
- Literal constructors that initialize `t` values from literal strings.
- Binary constructors that initialize `t` values from binary resources, holding preexisting instances; in the simplest case, simply copying the bytes in a v to initialize a v'.
- Pattern-based constructors that initialize `t` values from an aggregate data structure which may (but need not) include other `t` values. This would include building a list v' from a shorter-by-one list v by appending an element to the list, if that procedure is exposed as a (co-)constructor.
- Recursive constructors allow values obtained by pattern-based constructions to be "deconstructed" and used for recursive algorithms, as outlined earlier in the case of lists.

We could add further details to this breakdown—(co-)constructors that validate their input, for instance—but the basic idea is that types are often designed with implicit assumptions about how they are to be used, and these assumptions become manifest in what sorts of constructors are provided. These design patterns can be made more rigorous or explicit by consciously notating and classifying what sort of use-case is envisioned; one way to achieve this is by making object factories or co-constructors the basic public interface for a type, and then supplementing co-constructors with metadata that describes the type interface in a systematic manner.

Assuming this methodology, we then have an additional set of tools for reasoning about `t` values. Upon enumerating various *kinds* of (co-)constructors, as above, we can specify whether a `t`-value v could be the product of a co-constructor of a given kind, that is, whether v could be default-constructed, say, or constructed from a source-code literal. Intuitively, we then have an idea of "partitioning" the space of a type into regions based on the kind of construction that results in the corresponding `t`-values. This picture is hard to make fully rigorous, because it is not automatically given how we should think of type-instances as a "space." For some types, we can neatly list all their possible values (say, signed bytes are every number from `-127` to `127`), but in many varying-size types the actual set of values that could be represented at any moment, in any particular computing environment, will dynamically depend on factors such as available memory. It is impossible to say, for instance, just how long a list can become before it requires too much memory, which in turn would result in the proposed list failing to be constructed.

In short, we need analytic methodology that does not treat types as if they were "sets of values." In the framework of channels and carriers, we try to achieve this by reasoning about types through the carriers that hold type-instances, and by defining carrier *states* (including states orthogonal to any type system, e.g., a carrier which *doesn't* hold a value). With this foundation, we can talk about the "space" of type-instances in terms of *states* on carriers. Suppose a carrier `c` holds a `t` value produced by a co-constructor of a given kind ($\mathcal{K}$, for instance). We can then introduce $\mathcal{K}$ as a state on `c`: `c` is in the state of holding a value emerging from a $\mathcal{K}$-co-constructor. This provides potentially useful information. If $\mathcal{K}$ corresponds to a default-initialized "sentinel" value, i.e., a fallback like

null for unavailable data, then such a state corresponds to holding a conceptually "invalid" t-instance. Or, if $\mathcal{K}$ corresponds to a pattern-based construction suitable for recursion, v could be used in recursive algorithm.

Note also that many types have a notion of a "fallback" or "default" value, which may or may not be *valid*. For numeric types, that value is usually zero, but the meaning of 0 can depend on context. Consider a procedure that checks a database to learn someone's age: the default 0 may be intended to mean that this information was not found or not provided. However, in (say) a medical context, 0 may also be the (real) age of an infant. Analogously, an empty string might mean that someone's middle name is not known; or that someone does not *have* a middle name.

To avoid confusion, types often are built around "sentinel" null values, which cannot be confused with a conceptually meaningful value. In computer graphics, consider the case of color: a reasonable default value (for drawing a line, for instance) would be the color black, which also, in RGB encoding, corresponds to vector of three 0s, so it is consistent with default-to-zero conventions. On the other hand, a system may need to identify situations where a color is unknown or not specified (analogous to an unknown age vs. a baby's 0 years), so a type representing colors may have some extra value meaning "no color," which would not be confused with or deemed equal to black.

Analogously, some programming languages allow the (technically negative) number -1 to be used in an *unsigned* context, as a special "unknown" value. If someone's age is given as -1, then, it would be clear that the intent is to report that the age is not known, with no confusion vis-à-vis a child before their first birthday. Numerically, -1 would actually have a binary encoding (most likely) as **255**, which would never be confused with someone's real age. Similarly, types representing textual strings sometimes distinguish *empty* strings (like when someone is known not to have a middle name) from *null* strings (representing missing or unknown data).

In practice, accounting for cases of default or missing data is an essential part of designing types, qua digital artifacts. If a t value is not known or not provided, should a (valid) default value be used instead? Black, say, is a reasonable default for colors (though what about color-systems with transparency: should the default be fully opaque colors, with no transparency effects, or fully transparent and therefore invisible colors)? Conversely, 0 (when it also means zero years, not yet one year old) is probably not a reasonable default for someone's age. When data is missing, should default values be used; if not, how should the problem be represented? These decisions influence our conceptual understanding of types' spectrum of values and their expected uses. A default and conceptually valid value (like black in the realm of colors) is conceptually different than a default value which is *not* conceptually a "real-life" instance of the type (like -1 for someone's age)—let's call these *meaningful* and *meaningless* defaults, respectively—and that in turn is different from an invalid value that should generally not be passed between procedures (which we'll likewise call an *invalid* default). For an example of this last distinction, a sentinel "NaN" should rarely actually be passed to procedures expecting a number—on the premise that any calculation on NaN should yield NaN, so the call would be superfluous—while it is quite common to pass **null** pointers in C++, even though their conceptual meaning is that they do *not* point to any memory address (so, conceptually, they are not "real" pointers). In short, *black* is a meaningful default, null is a meaningless but not invalid default for pointers, and NaN is typically an invalid default for numbers—or at least this summarizes typical usage patterns.

Unpacking the conceptual variations between different type-instances—meaningful, meaningless, and invalid defaults, for instance, or literal-

initializable values—gives rise to a sense of types' "extensional space"; how there are (sometimes) patterns and groupings amongst type-values more specific than just being instances of their respective types. Arguably, such extensional groupings are to some degree analogous to Conceptual Spaces, marking the "space" of types' extensions in a conceptual fashion. Of course, extensions also have *statistical* distributions, which are a more precise fit for Conceptual Space Theory, but both of these models (we can call space-partitions based on criteria such as null or default-initialized values *pre-statistical*) are potential forums for integrating type theory with Conceptual Spaces.

5.1.5 Nonconstructive type theory

Thus far we have suggested that types' conceptual and operational profiles can be defined in part by describing the system of constructors (or co-constructors), through which their values may be created. The process by which a particular t-value v has been created can be a factor in how v may be used. For example, most functional programming languages allow procedures to be implemented via "pattern matching," which means splitting the procedure into different versions or different routines based on the nature of an input value, which can be determined by how it was constructed. A canonical example is procedures defined on lists: suppose v can be "deconstructed" into a smaller-by-one sublist v' and a single element e — v is v' appended by e. The right-hand side (v' with e) can be called a *construction pattern* which yields, or defines the provenance of, v. On that basis, a procedure that operates on v could equally well, at least logically or conceptually, be seen as operating on the v' and e pair. On the other hand, if v is an empty list, then algorithms need to proceed differently than for v non-empty. In combination, this yields an outline for procedures as follows: differentiate empty from non-empty v; for the latter, allow the procedure to operate on a v' and e pattern, rather than on v directly.

Programming languages that want to support these "pattern matching" features need two capabilities: they must implement procedures that bifurcate based on which pattern matches; and they have to take a multi-part structure as a procedural input, such as v' *and* e, in the place of a single carrier like v. One straightforward way to achieve this is by differentiating procedures based on the construction-patterns evinced by their arguments. For example, we can consider a procedure that *only* operates on *empty* lists, paired with a procedure that *only* operates on *non-empty* lists.

We then have to investigate how these distinctions intersect with the relevant type system. Should $\mathbb{T}$ stipulate that procedures for empty lists have a different *type* than procedures for non-empty lists? Note that in general the empty/non-empty distinction does not yield different type-attributions: a non-empty list is not a different *type* than an empty list (assuming compatibility in the type of elements declared to go in the list). There are nonetheless frameworks which would allow a *function* taking empty ts (for list-type t) being considered a different type than ones taking non-empty ts. For procedures taking non-empty lists, moreover, their argument can (potentially) be converted into a construction-pattern (like v' and e), so that the single input to (this version of) the procedure actually becomes two different inputs. Enabling procedures to be split and designed in this manner—split and paired off by pattern-matching and taking compound inputs—requires $\mathbb{T}$ types to be implemented with the requisite capabilities (e.g., calculating the proper construction-pattern for a given t-value). Because this sense of "pattern matching" is a common idiom in functional programming, it certainly needs to be accommodated in a broad-based type theory attuned to the

type systems of different kinds of formal languages.[7]

Conversely, though, it is just as common for types to *lack* the infrastructure for pattern-matching in this sense, so we need to have these features as *possible,* but not *necessary* aspects of types. For the sake of discussion, we will call types amenable to pattern-matching *constructive*; and, otherwise,*non-constructive*. There is no need here to formalize a technical definition of the comparison, but semi-formally we'll say that a *constructive* type has (or can be provisioned with) a procedure which, given any instance v, can return a construction-pattern that reciprocates the construction of v from other values. In this context we treat the functionality to calculate patterns as an ordinary procedural interface on a type: for constructive $\mathtt{t}$, there will be an associated "construction pattern" type (usually a type *different* from $\mathtt{t}$) and procedures to map $\mathtt{t}$s to their corresponding patterns. Pattern-matching can then be achieved, or at least emulated, by defining procedures on the construction-pattern types for $\mathtt{t}$, rather than $\mathtt{t}$ itself.

Implementing constructive types introduces some complications in conjunction with an overall type interface, which might not be immediately apparent. For instance, consider types that have (what we earlier called) "binary constructors," e.g., a $\mathtt{t}$ that can initialize values by copying the bit-pattern of some other $\mathtt{t}$-instance. For many $\mathbb{T}$, this option results in the theoretical possibility that $\mathtt{t}$s could be created with any bit-pattern whatsoever. In C++, for example, a pointer could potentially point to some random area of memory (after an error in neglecting to initialize the pointer, say), from which a dereferenced yields a $\mathtt{t}$ value manifest in a random sequence of bits. Such randomness is almost always an error, but this does not preclude code having to accept the possibility that $\mathtt{t}$-values might be "randomly" constructed. In this case, the data that could be used derive construction-patterns may well become corrupted.

Suppose a type offers an interface to return construction-patterns for all of its values, but makes assumptions about these values' binary layout when deriving those patterns. For instance, a list type might be based on a list-pointer together with fields indexing the start and end of the list. With this arrangement, one single list pointer, i.e. one list in memory, can be the basis for multiple $\mathtt{t}$-values, by varying their start and/or endpoints. A construction pattern for a list v could then be efficiently obtained by noting the element e at the start or end of v, and producing a v'-e pair by constructing v' as a variant on v with its start or end index advanced (respectively decreased) by one. This is a plausible "deconstructing" scheme, because all the intermediate list values obtained in construction-patterns, and then potentially used in recursive procedures, share the same underlying memory; there is no copying or modifying of in-memory data. A list is then considered *empty* if its start and end indices are the same. A recursive algorithm would work with a sequence of construction-patterns, with the two indices getting closer together, until the recursion would be broken by final list being empty.

The problem here is that this design only works if the start and end indices for the original v are in the correct order (the start must be less than or equal to the end). If that requirement cannot be guaranteed, then the above derivation of construction patterns cannot be guaranteed to work properly; in particular, a recursive algorithm might loop infinitely. As this example shows, a constructive type may need to double-check all values at their point of construction to ensure that the type's various fields and internal data are configured properly to support features such as pattern-matching. A constructive $\mathtt{t}$, in short, may need to ensure that no $\mathtt{t}$-values are constructed without certain validation checks

[7] Note that there are other, unrelated uses of the phrase "pattern matching" in programming and computer science, e.g., in the context of regular expressions, which is essentially entirely different from the current context.

being performed (this would be one use-case for a co-constructor). For instance, simply copying bit-patterns from one place to another may have to be disabled as a tactic for copy-constructing values, unless the newly constructed data structure is validated.

Given these considerations, it is certainly possible that some types will be non-constructive—i.e., they decline to provide procedures that return construction patterns, and to provision the support needed to use construction patterns—even if the logical characteristics of the types' values would seem to support a pattern-based interface. In practice, programming languages that enable pointer-based access to values, and pointer-dereferencing, tend not to natively recognize construction-patterns, and vice-versa.

The idea of proxying t-values via construction-patterns has mathematical foundations, emanating from "constructive" mathematics. In particular, consider "recursive" construction patterns, where a v is "deconstructable" into a v'/e pattern, and then v' is further substitutable by a further pattern based on a v'', and so on. In many cases, for any t-instance v, there is then a determinate sequence of constructions which eventually produces v, and likewise an "inverse" sequence of construction patterns that "undoes" those constructions. For instance, any list of numbers can be seen as the product of numerous constructions which begins with an empty list, and produces successively larger lists by appending one number at a time, eventually arriving at an end-result v. In a language such as Haskell, the notions of *list of elements* and *sequence of constructions (appending elements to a prior list)* are understood to be conceptually indistinguishable. That is to say, t values are understood to be internally inseparable from the progression of steps which provides a recipe for constructing those ts.

Mathematically, an analogous assumption is that the space of t-values is isomorphic to the space of construction sequences which yield ts. We can also say that the space of these sequences is a *model* for t. A type mathematically representing *lists of integers*, say—meaning in this context a logical specification of some mathematical space—can be said to be modeled by the space of programs that produce these lists by starting from an empty list and progressively appending numbers. This "space of programs" is a *model* for the type, insofar as it satisfies the types' logical requirements. This is one example of a project for analyzing mathematical spaces in terms of *finite constructions* that yield elements of those spaces. In constructive mathematics, proofs of propositions on such "finite constructions" is considered to be easier, or more logically sounder, than proofs that engage with infinite spaces and rely on logical indirections, such as the "law of the excluded middle."

Constructive types, then, inherit this mathematical backstory insofar as such types allow us to deem types' space of values as, in essence, logically indistinguishable from the space of construction-sequence that yield those values. A *constructive* type theory would be one that treats all types as constructive, perhaps on the basis of logical or philosophical reasoning: we can say in the abstract, for instance, that any list is *logically* isomorphic to a sequence of sublists building up to it. In practice, though, we propose considering a type constructive, only if it *explicitly* provides the infrastructure needed (or if that happens automatically given the relevant implementation language) so that "constructive mathematics" intuitions can be concretely leveraged. We will say that a type system is *non-constructive* if it does not *assume* that its types are constructive; a non-constructive $\mathbb{T}$ may still have *some* constructive types.

In short, a non-constructive $\mathbb{T}$ actually *generalizes* constructive frameworks: by allowing both constructive and non-constructive types, it presents a superset subsuming $\mathbb{T}$s that can model either non-constructive *or* constructive types or both. Non-constructive type systems, we contend, achieve the greatest breadth in covering the diversity of possible $\mathbb{T}$s, while stay-

ing within the bounds of software-centric, implementational rigor, especially in conjunction with the features of being *channelized*, and *cocyclic*, from which we derive the "NC4" moniker (**n**on-**c**onstructive, **c**hannelized, **c**o-**c**yclic). In the following chapters, we will implicitly adopt NC4 type systems as a foundation for studying the semantics of formal languages, in the hope of deriving a theory of synthesizing hypergraphs with Conceptual Spaces—which can be usefully paired with analogous unifications for *natural* language.

5.2 Types as conceptual structures

The types encountered in a type system $\mathbb{T}$ are technical artifacts, but in many cases they also are designed to model, or track information about, concepts in the everyday world: fragments of natural language (i.e., text); people; places; colors; events; medical procedures and diagnoses; demographic and government data; scientific data and research findings; and so forth. We can accordingly consider types (implemented in software components) as *conceptual* artifacts, but with the caveat that their conceptual details have to be modeled within the constraints of software environments and computational processes.

Most real-world-based types are syntheses of multiple dimensions, or "fields": a type representing a person, say, might include their name, date of birth, gender, marital status, address, phone numbers and other contact info, etc. Or, consider how we might lay out data for restaurant listings: GPS coordinates to locate restaurants on a map; street address; restaurants' names, hours of operation, and phone numbers; perhaps an indication of their relative priciness; perhaps a categorization of their style of cuisine (French, Italian, Chinese, Japanese ...). The type might also have certain "flags" with pieces of information in a yes/no format: do they take reservations; are they wheelchair accessible; do they accept credit cards; do they serve alcohol. An overall "restaurant" type (say, $\mathcal{r}$) would then be a "product" of these various fields. In Conceptual Space Theory, likewise, concepts are defined by a crossing of multiple dimensions, which collectively define the space of variations that their instances can occupy. Conceptual analysis can then proceed by isolating individual dimensions of variation before investigating how they unify to create our impressions of conceptual similarity and dissimilarity, how concepts are refined or generalized, etc.

In formal types, fields have different structural or extensional properties, which influence their conceptual status. One obvious criteria derives from the statistical distinction between *nominal*, *ordinal*, *interval*, and *ratio*. In describing a restaurant, say, the field representing "kind of cuisine" is presumably nominal, in that we identify certain terms, such as Italian, Japanese, etc., and assign the restaurant to one or another. This field probably has no "scale" or "metric"—French is not *more* or *less* than Chinese. A field representing *cost* may similarly be broken down into discrete options (inexpensive, moderate ...), but here there *is* a notion of scale (we can rank *moderate* as between *inexpensive* and *expensive*, say; and order restaurants from cheapest to costliest, or vice-versa). On the other hand, cost may also be figured by a metric, such as the average price of a typical meal, which would be a straightforward, increasing, whole-number scale; prices can be ordered and degrees of difference calculated: two restaurants are similarly expensive if their average meal costs roughly the same amount. Meanwhile, geospatial coordinates represent a two-dimensional and (up to approximation) continuous space, one that permits distances but not ordering, unless we are taking distance from one central point (e.g., someone looking for restaurants close to their home). Hours of operation, for their part, cannot obviously be "ordered", though we can determine if a restaurant is open at a particular time; we can also rank establish-

ments by how late they close, or how early they open; in principle, hours are cyclic, but when defining closing times, we just need to consider the window of hours during the evening and night (respectively morning and afternoon for opening times).

For a hypothetical restaurant $\mathcal{r}$ type, then, we can analyze their various fields in terms of their propensity for *ordering* and/or *distance*. We can say, that is, that some fields allow restaurants to be ranked in increasing or decreasing measures for some fields (e.g., average cost of a meal); and some fields permit the "difference" between restaurants, within the dimension of the field, to be quantified (average cost of a meal again, or location). Other fields allow for no particular comparison except for matching against single nominal values. Unless we impose some metric whereby, say, Chinese and Japanese restaurants are deemed more similar to each other than to French or Italian, the most we can say is that two Chinese (etc.) restaurants are likely to be considered similar by virtue of both serving Chinese cuisine. Still other fields allow for different kind of comparison if someone is looking for a restaurant meeting some criteria—that it accepts reservations, say, or is open at 10 p.m. on a weekday.

The statistical or qualitative structure of fields, along these lines, become implementationally significant if we seek to derive algorithms to match $\mathfrak{t}$-values to *queries* (say, to find a restaurant matching some customer's preferences), or to estimate whether $\mathfrak{t}$-values will be deemed similar or dissimilar (if someone likes one restaurant, an engine might look for "similar" restaurants to recommend). These requirements, then, translate to $\mathfrak{t}$-values as a whole: Can we quantify (perhaps by a single distance metric) the degree of difference, or similarity, between two values? Can we order any collection of $\mathfrak{t}$s and, if so, ranked by which dimension? Can we search a collection of $\mathfrak{t}$s and find a list of values that meet some search criteria? To put this last question differently, how can we define parameters for searching $\mathfrak{t}$ collections? Do we create a hypothetical $\mathfrak{t}$ —say, an Italian restaurant open at 10 p.m. that serves wine—and use that as a template for matching concrete values in the collection? Or do we create a different data structure, with a different type, aggregating search criteria against sets of $\mathfrak{t}$s? Should we, correlated with $\mathfrak{t}$, define a distinct type for searches yielding $\mathfrak{t}$s? In the case of restaurants, such a "search" type might specify a range of price-points, maximum geospatial distance from a central location, one or more kinds of cuisine, and so forth, replacing single fields in $\mathcal{r}$ with ranges and bounds.

In some ways, these operations of ordering, querying, and measuring difference/similarity are consistent with Conceptual Space Theory: they capture how conceptual reasoning can be bound to quantitative possibilities, using quantitative relations—distances, orderings—to reason through concepts' extensions. On the other hand, such quantitative reasoning does not constitute a full-fledged reduction of these concepts to quasi-mathematical spaces; it is not, say, that quantitative fields in a restaurant $\mathcal{r}$ type reveal how all conceptual details about restaurants can be reduced to numeric patterns. Instead, quantitative structures fall out as the result of *operations* on this type, operations like ordering, querying, or measuring similarity. We would argue that, as cognitive processes, sorting and comparing are more fundamental than construing relations numerically, although numeric patterns may arise organically in the manifestation of sorting/comparing cognitions. In short, the quantitative picture of (in this example) restaurants (or analogously we would say for many concepts) is derivative upon rational operations we perform *on* sets of concept-instances, more than latent mathematizations of an underlying conceptual space.

Analogously for formal types, many numeric structures come into play, not internally within those types' own fields or structures, but in terms of operations performed *on* types, and particularly on *collections* of $\mathfrak{t}$-instances, collec-

tions that can be ordered, queried, clustered by degrees of similarity, and so forth.

Suppose we have a restaurant database that tracks favorable reviews, assigning each restaurant a "grade" from, say, 0 to 100. Our $\mathscr{r}$ type thereby has another scalar field, which can be combined, say, with an average-cost-of-meal, yielding a two-dimensional space that restaurants can be mapped into. From the distribution of the resulting points, we could identify "good values," which are unusually highly reviewed for their price-point, or outliers in the opposite direction, where the review scores are lower than price would suggest.

In other words, a sample-space of restaurants mapped into the price/review space gives us a quantitative distribution, and our ability to compare restaurants in this way is doubtless a facet of restaurants' concept. But these quantitative details are only really salient when it comes to *comparing* restaurants, and statistically reviewing restaurant collections. The numeric structures are less conceptually foregrounded in our cognitive appraisals of any *single* restaurant. It is true that the quantified comparisons are possible via aspects that all restaurants have because of the their conceptual "package", so to speak; so mathematizable comparisons are latent in restaurants' internal conceptualizations. But these aspects only really become *quantitative* in the context of comparisons, whether these are explicit (e.g., analyses of a database) or more mental and informal. My judgment that a certain establishment is pricey, say, or cheap, inevitably results from a comparison (maybe subconscious) with other restaurants we have visited, or at least heard about.

The acknowledgment that quantitative structures thereby arise in the course of conceptualizing *restaurants* (in this case study) does not accordingly demonstrate that our conceptual activity representing restaurants as cognitive acquaintances—as a feature of the world we roughly understand, for which the concept serves as an orchestrative tool—is at some fundamental level essentially mathematical. Instead, numeric spaces and axes emerge from mental operations layered on top of our basic restaurant-cognitions, particularly insofar as our reasoning turns from thinking about individual restaurants to their comparisons. The analogous phenomenon in formal type theory would be that quantitative models of types' distributions come to the fore in conjunction with operations for sorting and comparing type-instances. We will now examine these kinds of operations in more detail.

5.2.1 Dimensional analysis and axiations

Let us assume we direct attention to types in a $\mathbb{T}$ which are characterized by numerous distinct fields, and also that we are interested in procedures for ranking and comparing $\mathfrak{t}$-instances. We can then analyze fields on the basis of how they might contribute to such comparative operations. We will use the phrase *intratype comparisons* to refer collectively to various ordering, comparing, querying, clusters, and measuring-dissimilarity operations.

A $\mathfrak{t}$'s fields can then be classified according to how they may contribute to intratype comparisons. In the abstract, such an analysis would be provisional, because certain type-implications may have their own peculiarities. For instance, it is reasonable to say that a restaurants' *name* is not a factor in estimating similarity: two restaurants with similar names are not especially likely to be similar in other respects. However, we can envision scenarios where textual similarity *would* be taken into account (e.g., a search engine trying to accommodate spelling errors). Or, consider the question we mentioned earlier as to whether factors such as Chinese/Japanese or French/Italian similarities (as styles of cuisine) should be modeled so as, in effect, to yield a distance metric on a nominal "kind of cuisine" dimension. These examples show that anticipating exactly how clustering or distance algorithms would be designed, in any concrete case,

is rather speculative. Nevertheless, we think it is possible to make some broad claims about how data fields *usually* work in the intertype-comparison context.

On that basis, then, we propose to distinguish five rough sorts of data fields, as follows:

1. Digital/internal fields: Computational artifacts, such as globally unique identifiers, which are employed by code managing type-instances behind the scenes but do not typically embody real-world concepts related to the type, and are not typically salient in intertype comparisons.
2. Textual fields: Natural language artifacts, such as names and descriptions, which would not normally be used directly for intertype comparisons. Ordering or measuring similarity on collections of textual contents tends to be difficult, or to have little actually conceptual resonance, unless some sort of natural language processing is used to extract structured data. For example, there is probably little conceptual significance attached to (dis)similarity or ordering among restaurant names, except perhaps if it is desired to list restaurants alphabetically (which in any case is a presentational matter more than a comparative one).
3. "Axiatropic" fields: We use this term to represent any fields that have nominal, statistical, scalar, or in any sense quantitative qualities that can be leveraged for comparisons. We include nominal fields (enumerations), because these are relevant to similarity—consider two restaurants both labeled "Chinese"—and also nominal dimensions can sometimes have extra comparative structure (e.g., an inexpensive-moderate-expensive scale is ordered by increasing cost). Other than enumerations, we define axiatropic fields as any dimension with numeric values where numeric properties are consequential for ordering or for measuring similarity, excluding, say, numeric ID's, where numbering has no structural meaning other than uniqueness, but including "locally" ordered scales, such as time points, as well as "globally" ordered dimensions, such as integer magnitudes (e.g., prices), and spaces that have no particular ordering, but which can be ordered via distances (like geospatial locations). Axiatropic fields can be seen as "axes" to which type-instances can be projected, and the union of ts axiatropic fields, which we propose to call ts *axiatropic structure*, defines a multi-dimensional space wherein ts are mapped to individual points. The tuple of values obtained from these fields we call an *axiatrope*, and the points to which a given t-axiatrope projects we call an *axiatropic image*. If desired, axes can be annotated with details such as valid ranges and units of measurement (consistent with, for instance, Conceptual Space Markup Language, to be discussed in later chapters).
4. Flags: We separate out Boolean values—along the lines of whether a restaurant is wheelchair accessible, or takes reservations—because these pieces of information are more likely to be used to match candidate values against criteria than for comparisons between values. This does not preclude some similarity algorithm from ranking, say, two restaurants that serve liquor, or take reservations, as a little more alike—i.e., these data points may add to a metric of similarity (or inversely subtract from a metric of difference) when they agree between instances, or contrariwise when they disagree. However, we would argue that conceptually these kinds of factors are more pertinent to building a sufficiently detailed picture of an instance than to intratype comparisons; on that premise we distinguish "flag fields" as a separate field grouping.
5. "Mereotropic" fields: These fields represent collections of values (lists, tuples, and so forth), rather than single values. In general, tuples are less conducive to direct comparison, without further analysis—say, compar-

ing two students' grades by comparing their average; or comparing two lists by counting the elements they have in common. Similarity metrics, then, can employ collections fields, but the calculations are more involved than just projecting type instances onto points in a mathematical space. We leave open the possibility that collections may have elements that are themselves collections, so that mereotropic fields can give rise to "mereological", or part-whole, hierarchical structures.

These different genre of fields are reflection in different compartments to a type's interface; to this list we would then add the portion of the interface related to constructing t-instances: constructors, co-constructors, and related functionality for testing the validity of a data structure and/or "deconstructing" values into a construction pattern. Collectively, we will refer to this last aspect—which does not involve fields *per se*—t's *constructive interface* or (with apologies if the neologisms are getting a little heavy-handed) its *nomotropic interface*. The fields (textual, flags, binary) which are neither axiatropic nor collections-based we will call *endotropic*. We will then call procedures related to restructuring or re-presenting t-values for analysis, GUI or visual rendering, serialization and deserialization, or database persistence, as collectively a *morphotropic* interface. We then have a partition of types' interface into five facets: nomotropic, axiatropic, morphotropic, endotrophic, mereotropic.

The *axiatropic* aspect of type-extensions may seem like a jargony designation of rather mundane statistical distributions of a set of values, since projecting data structures onto common axis-sets that permit quantitative comparisons is the bread-and-butter of data analytics. Such projects themselves certainly aren't a new theoretical posit. However, we wish to emphasize that in most types (especially ones modeling real-world objects) the process of setting up such statistical comparisons involves *filtering* fields so as to focus on statistically malleable information in each type-instance. It should be emphasized that the application of statistical metrics to value-sets constitutes a *projection* of each value onto a restricted axiated space that supports quantitative dimensions, whereas many types will *also* have qualitative data which is more difficult to accommodate numerically.

The fields that are likely relevant to comparing or querying ts are facets of t that need to be deliberately engineered. Implementers, in short, have to anticipate and provide procedures for client code—software using the type implementation—to interact with ts fields: iterating over collections, dynamically calculating views, querying against a prototype, ordering on one or another field, etc. These considerations inform the process of designing an *interface* for t. Overall, a type interface covers various tasks, such as constructing t instances in the first place, or integrating ts with other kinds of software components (serializing and deserializing ts for data sharing; sending ts to a database; showing ts in a GUI; etc.). We will use the term *paratropic interface* to that portion of a ts interface, which concerns sorting, comparing, and querying ts (or sets of ts).

In practice, the field-classification we proposed earlier would most likely be applicable to a type's "paratropic interface": it would help implementers reason about what data points to expose for a t and how precisely to set up the logistics for obtaining this data from t values. Taken in conjunction with the principle that formal types are (often) modeling artifacts that approximate real-world concepts, this discussion then suggests that such conceptual goals are mostly operative in the interface *to* types. That is, when considering how to use formal artifacts to proxy real-world, human concepts, the point is not only to assemble a list of particular details to keep track of (for a restaurant: name, location, cost, etc.). A type's effectiveness in representing human concepts is equally dependent on a programming interface that allows digital oper-

ations to be performed; operations which reflect how we conceptualize real-world phenomena, including by actions of ranking and comparing instances of the same concept.

We think this perspective is also consistent with natural-language: a conceptual account of *nouns*, for instance, should be oriented in the pragmatic employments of concepts as cognitive tools; the idea that we mentally *do things with* concepts. Perhaps there is a certain correlation between the idea that formal types' are conceptually defined by their *interface* and that, cognitively, concepts are constituted essentially by their intellectual-functional roles. We will explore "conceptual roles" further in Chapter 9.

5.3 Hypergraph ontologies

Hypergraphs—along with structures that can be modeled via hypergraphs, such as Conceptual Spaces—are an improvement on Semantic Web data formats and schema, addressing the limitations of a paradigm devoted to modeling complex information via first-order logic and non-nested graphs, with no notion of scoping or locality.[8] At the same time, a lot of effort has been extended building technologies to integrate heterogeneous data spaces via the Resource Description Framework (RDF) and RDF ontologies. In radiology, for example, attempts to better incorporate clinical and outcomes data are centered on ontologies such as RadLex, VISION, and SEDI, which we discussed in Chapter 3. As a result, the important consideration is how to employ hypergraphs as an extension to the Semantic Web when warranted, while preserving the virtues (and interoperating with) conventional RDF ontologies.

The idea that hypergraphs *extend* but do not *replace* RDF and the Semantic Web implies that hypergraph schema are extensions of (but not substitutes for) RDF ontologies, which in turn yields the notion of *hypergraph ontologies*. In a conventional RDF ontology, metadata is primarily associated with graph nodes and edges. In particular, nodes are referenced to uniform reference identifiers (URIs), such as web addresses, and edges are labeled with concepts formally defined in one or more RDF ontologies. Concepts which are used to annotate graph-edges, and which are given a fixed meaning in some controlled vocabulary, are often called "properties." One special "is-a" property is often used to connect nodes with concepts that classify entities into one of many categories defined in an ontology, often called "classes." As such, most RDF ontologies are primarily composed of *classes* and *properties*, each assigned a unique label. The purpose of metadata for a given graph is then to link nodes to classes (for example, specifying that one node represents a clinical trial and the second represents a patient), and furthermore to link edges with properties (for example, specifying that a patient-node is connected to a trial-node in that the patient is *enrolled in* the trial).

Hypergraph ontologies are similar to conventional RDF ontologies in that they likewise provide constraints and metadata for graphs. However, hypergraph ontologies are more complex, because hypergraphs are likewise more complex than ordinary graphs. In particular, hypergraphs have different layers of structure: whereas RDF nodes are intended to represent a single concept or value (such as a number, date, personal name, or URL) a *hypernode*, within a graph, typically encompasses multiple pieces of information inside it (often called *hyponodes*, *projections*, *inner elements*, *roles*, or just *nodes*).[9] In general, when analyzing hypergraphs it is

[8] These are familiar critiques, but laid out with particular thoroughness by the conceptual space community, as we will discuss in Chapter 9.

[9] The term "roles" is used by Grakn.ai (see https://dev.grakn.ai/docs/schema/concepts); "projections" is used by HyperGraphDB (see http://www.hypergraphdb.org/?project=hypergraphdb&page=RefCustomTypes); "inner entity" is used by the biointelligence project [7], where the corresponding notion of "external entity" refers to what in

necessary to distinguish at least two "tiers" of nodes, *hypernodes* and *hyponodes*, such that hyponodes are contained within hypernodes. As a result, hypergraph ontologies need a corresponding distinction for node and edge annotations: insofar as nodes are categorized via classes, and edges via properties, it is necessary to stipulate whether these classifications apply to hypernodes, hyponodes, or some combination of the two.

A further complication arises because, even though hypergraphs represent nested or hierarchical structures, these hierarchies are often partial or overlapping. For example, a patient is *part of* a clinical trial, but a patient is also included in other collections as well; for instance, a patient may be enrolled in a specific health plan (for insurance coverage). One technique for modeling overlapping hierarchical data via hypergraphs is to employ "proxies," which are digital identifiers encoding a multi-faceted concept into a single value that can be part of a hypernode (proxies are similar to "foreign keys" in SQL). Therefore each patient, represented by its own hypernode, has an identifier that can be a proxy-value for the patient; for example, a value assigned to a hyponode becomes included in the hypernode encoding the list of patients enrolled in a clinical trial, or in the hypernode encoding the list of patients enrolled in a specific health plan. Hypernodes can then be linked to other hypernodes by virtue of proxies (e.g., the trial-to-patient connection), and also by virtue of overlap (e.g., the set of all patients both enrolled in a given clinical trial *and* enrolled in a given health plan).

In sum, compared to RDF—where there is one single sort of node-to-node relationship, based on whether or not an edge exists between nodes and how this edge is labeled—hypergraphs are more flexible/expressive, because they have multiple genres of node-to-node relationships: the relation between hypernodes and their inner hyponodes; between hypernodes and one another; between hyponodes in different hypernodes; and variations on each of these relation-types, wherein relations are defined indirectly through proxies. Moreover, in addition to hypernodes and hyponodes, hypergraphs afford additional levels of detail, such as *frames*, *channels*, and *axiations*. All of these details provide different "sites" where hypergraph annotations and metadata may be defined.[10]

An additional distinction within the Semantic Web is the contrast between *reference ontologies* and *application ontologies*. In general, *reference ontologies* are general-purpose schema intended to establish conventions shared by many different applications, to ensure that a large collection of data-producing software in a given domain is interoperable. By contrast, *application ontologies* are narrower in scope, because they are more tightly integrated into applications that directly send and receive data. Ontologies of either variety are used by software to interoperate with other software: so long as two applications are using the same ontologies, it is possible to ensure that one application can understand the data produced by a second, and vice-versa. However, such inter-operability is only potential; it is the responsibility of programmers to actually implement code which produces and/or consumes data that conforms to the relevant ontology specifications. In general, application ontologies are structured in such a way that these concrete implementations are more straightforward to produce, compared with reference ontologies. Reference ontologies offer little guidance to developers vis-à-vis how to directly support the ontology within application code. Conversely, application ontologies more rigorously

other contexts might be called other hypernodes linked to a given hypernode via hyperedges.

[10]This means that formats for describing hypergraph ontologies have to be more expressive than RDF ontologies, because RDF ontologies need only to classify metadata as node-annotations or edge-annotations; by contrast, hypergraph ontologies need to distribute annotations among multiple sites of graph structure.

describe the data structures that applications must implement to properly manipulate data that is structured according to the specifications of the ontology.

Within data mining and image analysis, hypergraph models are used in different ways for different algorithms. In the context of COVID-19 radiology, [6] (which we cited last chapter) describes an algorithm for assessing the probability of SARS-CoV-2 infection from chest CT scans, where hypernodes represent high-dimensional vectors (191 dimensions overall) and hyperedges represent k-nearest-neighbors; here each hypernode represents an entire image, mapped to a 191-dimensional feature-vector. In contrast, other image-analysis methods use hypernodes to designate smaller segments *within* the image, where hyperedges designate geometric adjacency and/or feature-space similarities. Whatever the algorithm, hypergraph analyses would employ a hypergraph library to store preliminary data for analysis and/or for serialization within a data set. One benefit of a hypergraph application ontology is therefore that these data structures used internally to implement analytic methods can be directly expressed within the ontology, whereas RDF ontologies can only model hypergraphs indirectly.

Although it is theoretically possible to encode data directly via RDF graphs, it is far more common for applications to employ tabular and/or hierarchical formats for data sharing, such as spreadsheets, protocol buffers, XML, or JSON. As a result, the role of ontologies for constraining data structures (so that they adhere to common standards) is indirect. It is useful to remember that ontologies are, at their most basic level, controlled vocabularies; as such, ontology constraints often amount to stipulating a set of acceptable terms for a data value, column header, or annotation. For a trivial example, our calendar recognizes 12 month names and 7 day names, which constrain the set of values permissible for "month" and "day" within a calendar date. These terms are so commonplace that a "date ontology" is unnecessary, but in scientific or technical domains, it becomes necessary to define vocabularies of allowable names or labels for specific data fields that representing some scientific value or measurement. For instance, the Ontology of Vaccine Adverse Events [8] provides a nomenclature for use in adverse events reporting, so that researchers or clinicians can describe symptoms via canonically recognized terms, rather than through informal text descriptions. In general, ontologies constrain data sets by stipulating that particular individual values within the overall data collection have names or descriptions whose associated set of possible values is prescribed *a priori* by the applicable ontology. However, the relationship between ontologies and concrete data sets must itself be documented, which is where application ontologies can become relevant: application ontologies provide a bridge between reference ontologies and the applications which use them (along with the data generated and shared by those applications).

To preserve the benefits of RDF ontologies—while also addressing those lacunae that make the Semantic Web "not (really) semantic"—hypergraph ontologies need to model constraint schema on hypergraph constructions (which have significantly more parameters of structuration than RDF graphs), while also connecting these schemas to the controlled vocabularies and logical axioms of Semantic Web (particularly OWL) ontologies. There are as such several areas of detail within hypergraphs where links to RDF ontologies may be drawn, which are outlined here[11]:

Hypernodes' Cocyclic Type Structure One of the central principles of hypergraph data modeling is the use of *hypernode types* to specify what sort of information is necessarily associated with a hypernode. In particular, a hyper-

[11] A full explanation of these concepts and terminology depends on an in-depth treatment of hypergraph type theory, which is outside the scope of this proposal.

node encompasses multiple hyponodes, each with their own type. These hyponodes represent information in some sense "contained within" or "tightly connected to" a hypernode (whereas data less canonically associated with each hypernode would, in general, be asserted via hyperedges, rather than via hypernode/hyponode containment). To ensure that hypergraphs are predictably organized, hypernodes cannot have arbitrary collections of hyponodes, but must instead be aggregates of hyponodes, which are assembled according to a schematic pattern, defined in terms of hyponode types. For maximum generality, a hypergraph type system should allow hyponode-type patterns to be as flexible as possible without introducing a need for metadata asserted at the level of individual hypernodes, rather than hypernode types; this motivates the idea we proposed above of a "cocyclic" type system, which is minimally constrained (but not unconstrained).[12] When translating RDF ontologies to hypergraph schema, accordingly, one consideration is whether edge-requirements are sufficiently ubiquitous in some context (e.g., with respect to some **rdf:class**) that these edges should be translated to hypernode/hyponode inclusions, and then to define a pattern of hyponode types for the corresponding hypernode type (the alternative is to retain the edges as links between *hyper*-nodes if they are *sometimes* but not *always* joined.

Roles, Projections, and Dimensional Annotations A hypernode type provides a schema defining a sequence of hyponode types; it is sometimes said that the hypernode "projects onto" that space of hyponode types. This projection is minimally characterized by hyponode types, but some hypergraph systems allow the projection to be *annotated*, introducing additional metadata that constrain (or augment the expressive power of) the enclosing hypergraph. Annotations can define scales/units/levels of measurement, probability distributions, situational roles, and other details lending semantic grounding to the data-field encapsulated by a hyponode. This metadata can then be a vehicle for translating RDF class constraints to hypergraph schema.

Semantic Nominal Dimensions The most direct translation of controlled vocabularies to a hypergraph context is often that of constraining the space of variation for one specific projection to a set of allowable terms. In the typical case, a hyponode type encapsulates a nominal set of values (i.e., an enumeration), so any hypernode including that type as one of its projections, is constrained by the labels registered in the vocabulary (a related formulation replaces non-hierarchical vocabularies with *taxonomies*, where some labels are treated as more or less granular variants of others).

Dimension Aggregates, Domains, and Conceptual Spaces Conceptual Spaces can be modeled in the hypergraph context by noting that hyponode projections are sometimes interdependent: dimensions tend to aggregate into semantically related groups (such as *latitude* and *longitude* as geographic markers). In Conceptual Space models, accordingly, projections are split into two levels—*dimensions* and *domains*—while other dimensional-analytic constructions (such as scales and units of measurement) are carried over (see [1]). Conceptual Space Theory also then introduces concepts of fuzzy logic or "convexivity" (according to different metrics) to simulate patterns in human conceptualization. We will discuss Conceptual Spaces in greater detail over the next few chapters.

[12] Because a pattern of hyponode types is "cocyclic" if the type-sequence includes a (possibly empty) tuple of types with no requisite pattern (called the "precycle") followed by a repeatable type-tuple, cocyclically typed hypernodes can represent expandable data structures, such as lists, stacks, queues, deques, and dictionaries. A typical hypernode type may then indirectly include multiple collections-types via proxies.

Probabilistic, Temporal, and Overlapping Hypergraphs Other forms of metadata constrained via ontologies can be expressed in terms of annotations defining weights or probabilities on hypernodes and/or hyperedges. One example is the juxtaposition of alternative markup hierarchies, in the context of hypergraph representations for Concurrent Markup languages, such as TAGML [2]. Numeric edge-annotation can represent weights (e.g., provide measures of the degree of uncertainty in the edge's relation actually obtaining), but constructions similar to weights have other sorts of applications. For instance, edge-annotations can be measures of time-spans, allowing hypergraphs to describe "entity-event models."[13]

Proxies, Inverted Proxies, and Double-Inverted-Proxy Constructions As described earlier, hypernodes can assert "containment" of other hypernodes by containing a *hyponode*, which *proxies* the second hypernode. An *inverted proxy* connection is therefore the mirror-image of this assertion (which may or may not be formally recognized by the hypergraph). A *double-inverted-proxy* connection is accordingly the relation obtaining between two hypernodes which are both proxied by one third hypernode (using the earlier example of proxies, the fact that two patients are enrolled in the same clinical trial). Many graph connections identified in a Semantic Web context (e.g., by SPARQL queries) are likely to be translated to double-inverted proxies in a hypergraph context.

In general, these hypergraph constructions represent sites for asserting constraints that (for RDF ontologies) would be defined on classes or properties; they are therefore a natural scaffolding for translating RDF ontologies to hypergraphs. Such a translation mechanism allows existing ontologies—which may play a valuable role in specifying protocols for workflows and data sharing between software components—to be reused in a hypergraph modeling environment.

As illustrated by CAPTK (discussed in Chapter 2), multi-application workflows are characterized both by the data that is transferred between applications and by the operations that connect the two applications, that is, the procedures enacted by each application when they become operationally linked. As a preliminary model, we can identify two stages of operational connection between an already-running application (which may be called the *primary* component) and a second application launched by the primary (which may be called the *peer* component). The first stage occurs when the primary component launches the peer component, and is characterized by two operational sequences: procedures enacted by the primary prior to this launch, and procedures enacted by the peer subsequent to the launch. A second stage occurs when the peer component has completed its actions, and sends data back to the primary component, which again involves two operational sequences: procedures enacted by the peer prior to the transfer, and procedures enacted by the primary subsequent to the transfer. Fully describing the procedural workflow therefore entails specifying four operational sequences: primary, then peer, during the launch stage; and peer, then primary, during the transfer stage. A schema describing the operations performed during these four sequences can be called a *procedural ontology*.

Consequently, rigorous models of multi-application networks should *synthesize* information about data structures (the type of information shared between application-points) with information about procedural workflows (describing operational sequences prior to the launch and transfer stages of a multi-application linkage). The synthesis of this structural and procedural information can be called a *procedural application ontology*.

[13] See https://allegrograph.com/consulting/entity-event-knowledge-graphs/.

One area where "procedural" Ontologies would differ from conventional Ontologies on the Semantic Web is that Ontologies in the former sense seek to characterize procedures (which by nature are digital artifacts) as well as objects (whether concrete or abstract). Of course, the Semantic Web notion of "object" is highly general, and certainly encompasses a broad range of abstract entities, including anything existing in the context of computer software and source code. So procedures as objects—or the analogous equivalence in the programming context, "functions as values"—can certainly be modeled with some degree of precision in a Semantic Web context. The difference between Semantic Web and "hypergraph" Ontologies is not absolute, but it involves shades or degrees of emphasis. In the context of hypergraph Ontologies, we can say that an *intrinsic* detail of computational data structures is that many data-types are collections of values which instantiate other data types—that is, containers that contain multiple instances of other types, and these containers can vary in size as values are added or removed. Different types of containers add or remove values in different ways: for instance, many containers are ordered, and restricted so that new values can only be placed at the end (or, alternatively, at the beginning) of the list. For these sorts of data structures, modeling procedures governing how the structures may change state (e.g., how lists may acquire new values) is an intrinsic dimension of modeling the semantics of the data structures themselves. This interconnectedness between data structures and procedural requirements is not entirely outside the representational scope of the Semantic Web, but nor is it the primary conceptual focus of Semantic Web Ontologies. However, the interconnectedness of data structures and the procedures that operate on them *is* an intrinsic concern of programming language type theory. A thorough foundation for reasoning about data sharing therefore calls for some form of hybrid analysis encompassing both Semantic Web Ontologies and type theory, as we will briefly review in the next subsection.

5.3.1 Type theoretic foundations for hypergraph-based data sharing

Data communications is a trade-off between optimization for space and bandwidth (often it is important to limit the number of bytes in each unit of information to a minimum), security, and programmability (networks that aim to support many developers spread across multiple projects and institutions need to be more open-ended than those centrally controlled by a single team). Ideally, then, each data type and data structure will be amenable to serialization in multiple formats, which each elevate one priority or another; XML and JSON are more flexible in an open-ended project, for example, whereas binary serialization is more efficient. Software models should be rigorous: there should be documentation of reasonable values for data fields, and (where applicable) their units of measurement. Preferably there should be some automated testing and/or type-checking to ensure that these expectations are sustained. Such specificity allows programmers to maximize resources that in some contexts may be restricted, such as internet bandwidth. On the other hand, requirements engineering should not degrade production performance, so verification should be executed in a special run mode that can be disabled for deployment, when necessary. For speed optimization, data types often fall back to types, such as 4-byte integers, which can support many more values than actually are conceptually meaningful. In these situations, the proper solution is not to degrade runtime performance with smaller and/or more complex types, but to model those types as predeployment checks of some sort, such as debug assertions testing the success of a type cast.

Similarly, conceptual modeling needs to anticipate the diverse ways that people express

concepts. For instance, someone's age is often measured in years, but with respect to toddlers, many people cite an age in weeks or months. If age is extracted from textual input, it is necessary to anticipate phrases such as "six weeks" or "six months"; potentially this should be reflected in the types through which age is modeled. If someone enters a child's age as six months, it might be disorienting to see this subsequently listed as, say, "zero years." This is an example of how provisional assumptions about the proper type representation of a human concept can be too simplistic, and therefore degrade user experience. It also shows the benefit of a rich type system, where types can be crafted to better model human concepts: if ages such as "six months" are recognized, then functions for comparing and binary encoding ages need to be implemented accordingly. A key role of a type system is logically organizing functions necessary for the type to work properly, which becomes more necessary as types acquire greater complexity (in the form of special flags and values) to adapt to human concepts and/or to interfacing with speech and natural language.

The above examples show why semantic modeling can be important to software development: semantic models are guidelines that improve applications' robustness and correctness by simultaneously codifying developers' and stakeholders' expectations and anticipating concepts which end users will bring to bear on their interactions with deployed software. Unfortunately, there are several different notions of semantic models, with roots in distinct academic fields, which do not precisely overlap, making it hard to integrate multiple semantic models into coherent, multi-paradigms information systems. Among semantic modeling paradigms, we can perhaps identify three that are particularly influential: grammars and lexical specifications for natural language processing; formal Ontologies described in formats such as OWL (web ontology language) and providing semantics for data formats, query systems, and persistence mechanisms, such as RDF, SPARQL, and graph databases or "triple-stores"; and the formal semantics at the foundation of type systems and implementations of concrete programming languages (implementations of interpreters, compilers, and/or runtimes).

In many modern-day applications, all three of these paradigms will likely play a role. For example, in a medical setting, bioinformatic semantics and communications are often modeled by some variation of OWL and RDF; patients' natural language (e.g., in the form of clinical narratives) is an essential part of thorough health records; and application-specific data types need to encapsulate aggregates of medical information as communicated between applications via protocols such as DICOM (Digital Imaging and Communications in Medicine) and HL7 (health level seven international). It is also important to observe that document-level criteria (like restrictions on XML documents) do not provide enough behavioral specification to address operational concerns, especially in domains requiring complex calculations and native code libraries (medical imaging, robotics, 3D graphics, image analysis, video analysis, or simulations of physical systems). Robust technology in these environments demands that a given real-world concept is modeled by behaviorally equivalent types at different points on a network, and that a serialization format (such as XML) serves merely as a conduit for typed values between network points. The components that send and receive typed values should be designed and tested for conformance to behavioral expectations once they have a fully formed instance of the relevant type in running memory; this is a more rigorous test than can be applied by checking an XML document against a Document Type Definition, for example. It seems fair to say the internet technology often fails this standard, with a frequent use of XML or JSON as a raw data stream that is processed as an untyped data structure on the receiving end, rather than

packaged into a type or class whose overall behavioral properties are well-documented.

Expressing operational constraints through languages' type systems and implementational mechanisms can be difficult. Consider the example of using a non-constant reference parameter passed to a function so as to initialize a value. This design requires that the function body always assign a proper value for that parameter and never attempt to read its value prior to assignment. This rather straightforward requirement is hard to express in many languages; in a pure-functional context the use of non-constant reference parameters is not directly allowed in the first place (forcing a more complex syntax at the call site), whereas in an Object-Oriented or procedural language, there may be no distinct type to capture a guarantee of initialization (analogous to how a CONST assertion guarantees that a value is "not" changed). Type systems are often designed around optimizations which may be useful in many contexts, but which are not always realistic, and the attempt to guide programmers toward writing optimizable code can unnecessarily obscure code that needs to be structured differently: compare the syntax for monads and strict evaluation in Haskell against the (simpler and more transparent) syntax for lazy evaluation, which is paradigmatically preferred. Conversely, expressing lazy evaluation in an object-oriented language such as C++, is convoluted and requires library support. These trends illustrate how type systems emerge from (sometimes too abstract) mathematical theories and need to be supplemented by semantic models that recognize conceptual as well as mathematical formalisms.

The consequence of this discussion in the present context concerns the problem of meta-modeling: the preferred frameworks wherein specific concrete data models may be developed. Various data structures—such as hierarchical trees or Semantic Web style graphs—have been proposed as universal meta-models, with the idea that such representations are sufficiently abstract and flexible that they can accommodate any data-modeling requirements in a computational/digital environment. This section, however, has attempted to identify reasons why these general-purpose data structures are suboptimal for data modeling at the level of requirements engineering and detailed procedural analysis. In this book we claim that hypergraphs, augmented by notions taken from Conceptual Space Theory, can provide a better meta-modeling foundation.

5.3.2 Hypergraphs as a meta-model for data sharing

Publishers, in recent years, have begun to encourage authors to share their research data, and to incorporate "Data Availability" or "Supplemental Materials" sections as an integral part of their journal and book publications. Initiatives such as FAIRSHARING or the Bill and Melinda Gates Foundation Guidelines for Authors (which is part of FAIRSHARING) provide recommendations to help authors produce data sets that are Findable, Accessible, Interoperable, and Reusable (FAIR). Such initiatives represent a step toward realizing the goal of engineering a broad ecosystem of open-access scientific data, an ecosystem that interconnects and interoperates with both scientific publications and scientific-computing software applications. However, this goal is hard to attain, because—in reality—scientific applications, publishing software, and data-hosting technology remain largely siloed from one another.

The FAIRSHARING standards also include several dozen domain-specific guidelines collectively referred to as MIBBI (minimum information for biological and biomedical investigations). The MIBBI specifications serve as a checklist for authors preparing data sets to ensure that they properly, and in sufficient detail, document their research and/or laboratory protocols (there are also research-documentation formats associated with specific publishing plat-

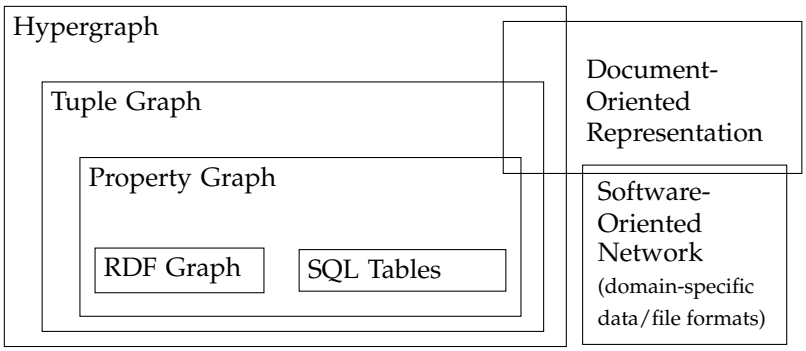


FIGURE 5.1 Relationships between different kinds of data models.

forms, such as *Springer protocols*, part of Springer Nature). As these examples demonstrate, researchers have numerous options for how they may document their research data. However, none of these documentation format options are closely integrated either with publishing software—in particular, with software used to compose books/articles for publication—nor with scientific software applications used to acquire, analyze, and visualize research data.

Data-sharing in the publishing context is often based on the Semantic Web. For example, "Research Objects," a standard model for composing reusable data sets, relies on the canonical Semantic Web data format, RDF (resource description framework), to encode research metadata. Semantic Web data is often criticized, however, for being "flat"—that is, RDF does not directly recognize any information structures that involve multiple levels or organization, or contexts: data collections (such as unordered sets or ordered lists), clusters of interrelated pieces of information, contexts where certain connections between information units have elevated significance, and so forth. To redress these limitations, multi-tier models such as property graphs and hypergraphs have been proposed as alternative vehicles for Semantic Web data. Though property graph and hypergraph database engines are increasingly popular in business IT, these technologies have not been comparably adopted within the publishing industry.

The interrelationships between several different data-modeling paradigms—and hybrid models that seek to unify multiple paradigms—are sketched out in Fig. 5.1. "Document-oriented" architecture in this context refers to formats such as XML, which allow arbitrarily nested content; these formats can model information with multiple levels of organization, but they are often difficult to work with in text-mining and data-mining contexts. Property graphs augment the flat-graph model by allowing sets of properties to be defined on edges and vertices. Property graphs, in turn, may be generalized to "tuple" models (where nodes could be collections of other nodes), and then subsequently to hypergraphs proper, which can be seen as tuple-graphs augmented with extra details characterizing the information encompassed by individual nodes. As one proceeds from more restricted to more flexible data models, we achieve capabilities to represent larger classes of data structures, so that general-purpose data-sharing initiatives should, in principle, embrace broader frameworks (such as property or tuple graphs or hypergraphs proper) in lieu of narrower ones (such as SQL or RDF). However, it is a challenging problem to implement programming environments for the more complex graph models—in particular, encoding, querying, and validat-

ing complex graphs—which may be one reason why such technology has not been embraced by publishers, despite there being promising code libraries available that could serve as a foundation for technologies targeted to authors and publishers.

Overall, in conclusion, hypergraphs represent a logical evolution of different meta-modeling paradigms, and they subsume many other data structures as special types or cases. Accordingly, we propose intuitively that hypergraphs incorporate the desired structural features of numerous other paradigms, while also introducing added structural details that can make hypergraphs a preferred foundation for database engineering and data-sharing. Of course, this intuition derives from a rather cursory overview of competing data-modeling paradigms. We will develop a more substantial theory to back up these ideas in subsequent chapters.

References

[1] Benjamin Adams, Martin Raubal, A metric conceptual space algebra, in: International Conference on Spatial Information Theory, Proceedings, 2009, pp. 51–68, https://pdfs.semanticscholar.org/521a/cbab9658df27acd9f40bba2b9445f75d681c.pdf.

[2] Elli Bleeker, et al., Between flexibility and universality: combining TAGML and XML to enhance the modeling of cultural heritage text, http://ceur-ws.org/Vol-2723/short39.pdf.

[3] Nathaniel Christen, Hypergraph type theory for specifications-conformant code and generalized lambda calculus, in: Amy Neustein (Ed.), Advances in Ubiquitous Computing: Cyber-Physical Systems, Smart Cities, and Ecological Monitoring, Elsevier, 2019.

[4] Bob Coecke, et al., Interacting conceptual spaces I: grammatical composition of concepts, Extended version of Proceedings of the 2016 Workshop on Semantic Spaces at the Intersection of NLP, Physics and Cognitive Science, pp. 11–19, https://arxiv.org/pdf/1703.08314.pdf.

[5] Ivan Čukić, Functional and Imperative Reactive Programming Based on a Generalization of the Continuation Monad in the C++ Programming Language, Dissertation, University of Belgrade, 2018, http://www.math.rs/files/ivan-cukic-phd.pdf.

[6] Donglin Di, et al., Hypergraph learning for identification of COVID-19 with CT imaging, Medical Image Analysis 68 (2021), https://pubmed.ncbi.nlm.nih.gov/33285483.

[7] Toni Farley, et al., The BioIntelligence framework: a new computational platform for biomedical knowledge computing, Journal of the American Medical Informatics Association (JAMIA) 20 (1) (2013) 128–133, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3555311.

[8] Erica Marcos, et al., The ontology of vaccine adverse events (OVAE) and its usage in representing and analyzing adverse events associated with US-licensed human vaccines, Journal of Biomedical Semantics 4 (2013), https://jbiomedsem.biomedcentral.com/articles/10.1186/2041-1480-4-40.

This page intentionally left blank

CHAPTER

6

Using code models to instantiate data models

OUTLINE

6.1 Introduction

In their synthesis of hypergraph categories and Conceptual Space theory that we reviewed in earlier chapters, Coecke *et al.* [9] adopted an approach for *syntax* (based on hypergraph categories) that is familiar in a computer-science setting, while favoring for *semantics* a paradigm (Conceptual Spaces) which emerged from a linguistic context. Category theory is not without precedent in linguistics, and likewise some projects have attempted to formalize Conceptual Spaces for computational applications, such as data modeling [1], [2], [45], [17], [56] and AI [16], [5], [20], [21], [31], [32], [53]. Nevertheless, Coecke *et al.*'s hypergraph-and-conceptual-space approach, rooted in Quantum NLP—natural language processing carried out on Quantum

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00012-X

computers (or simulations thereof engineered for research purposes) [36], [37], [15], [33], [41], [47]—represents a hybrid paradigm integrating perspectives from both formal analyses of computations and programming languages and from natural language.

Correlations between formal and natural languages are of particular interest to NLP—because NLP by definition needs to use computers, equipped only with formal algorithmic capabilities, to understand (often ambiguous or context-sensitive) natural language. Conversely, formal programming language theory (e.g., software language engineering) may draw ideas from natural language on the premise that (although first and foremost grounded on the affordances and limitations of digital artifacts such as parsers and compilers) programming languages are designed by people, where (at least for high-level languages, as compared to machine and assembly code) human readability and understandability is an important aspect of well-written code. This chapter will focus on programming languages, not natural language, but the use of hypergraph categories to structure the framing of NLP problems in terms suitable for Quantum processors (with quantum gates, qubits, and so forth) will serve as a motivating example for structures we present as a generalization on hypergraph categories.

In the case of Coecke *et al.*, the authors emphasize that syntax and semantics should mirror one another so that dynamic processes in both areas can be linked together (specifically, in category-theoretic terms). In their words: "[T]he general programme is [to] interpret the compositional structure of the grammar category in the semantics category via a functor preserving the type reduction structure .... This functor maps type reductions in the grammar category onto algorithms for composing meanings in the semantics category" (page 2). Adapted to conceptual spaces, their "programme" can be more rigorously laid out: "[W]e construct a new categorical setting for interpreting meanings which respects the important convex structure emphasized in Conceptual Spaces theory. We show that this category has the necessary abstract structure required by categorical compositional models. We then construct convex spaces for interpreting the types for nouns, adjective and verbs. Finally, this allows us to use the reductions of the pregroup grammar to compose meanings in conceptual spaces" (pages 2–3). A core specification here is that *type reductions in the in the grammar category* map onto *algorithms for composing meanings.*

It is obvious that "meaning is compositional" (a paradigm so entrenched, or perhaps so self-evident, that Jerry Fodor could teasingly insinuate its *a priori* resolving all questions about whether thought precedes language or vice-versa [13]). The notion of compositionality is more substantive in the context of "syntax-semantic interface," or specifically questions about how syntax and semantics respectively contribute to linguistic composition [30], [10], [25]. Coecke *et al.* make the further implicit claim that the syntax-semantics interface can be formally modeled via Category Theory: semantic composition is driven by compositions on the syntactic level with a force and causal precision that can be expressed mathematically, as if semantic constructions are transpositions or translations of syntactic constructions, akin to mapping a set onto its image under a mathematical operator.[1] We can then ask what *causes* syntactic constructions to engender semantic ones.

[1]To be sure, almost every NLP method assumes that syntactic and semantic construction-patterns can be tightly integrated, because this is precisely what can make semantics tractable to computer algorithms. The *issues* at stake in this assumption—first *whether* quasi-mathematical treatments of semantic constructions being the map-image of syntactic constructions is sufficiently true for natural language, without oversimplifying linguistic nuance, and second *how* syntax and semantics are integrated in this manner—can be tightly expressed by adopting terms and perspectives from construction grammar [14], [38], [55]. Specifically, we can say that semantic constructions (or *paradigmatic* constructions, alluding to the paradigmatic/syntagmatic opposition) are

As will be analyzed more thoroughly in Chapter 9, we take the perspective here that semantic constructions have *more information content* than their component parts, and that syntactic constructions thereby derive causative force from their encoding rules for how semantic elements can be unified with a specific emphasis on *increasing* information content.[2]

We can, in short, picture semantics as a kind of *path*, which leads in a direction of more information, more detail, and more specificity, wherein the terminus of that path is a sentence's overall complete idea (e.g., the proposition which a sentence conveys, if it is expressed in illocutionary terms—as an assertion, rather than a question, request, expression of desire, and so forth).[3] Bob Coecke implies a similar information-theoretic model when he suggests that syntactic relations "change" words (viz., the cognitions they tokenize; sentences "update their meanings"): "A sentence is not a state, but a process, that represents how words in it are updated by that sentence" [8, page 18]. Coecke *et al.*'s category-theoretic machinery serves (without using these specific terms) to specify the nature of information-augmentation paths via morphisms at the syntactic level (providing "steps" toward a sentence's total information content), which engender information-bearing constructions at the semantic level. The phenomenon of semantic constructions being reflected images of syntactic ones can be accounted for, on this perspective, by treating semantic content as some body of information that is built in stages, thereby implying a path *leading* along a gradient of information augmentation to the construction in its totality; such paths are causally grounded in the corresponding syntactic construction, and, on this theory, these paths thereby form the causative nexus of the syntax-semantics interface.

For reasons to be clarified in Chapter 9, we will use the term *syntagmatic constructions* to refer to models of syntactic construction that are explicitly based on this notion of information-content amplification. Whereas syntactic rules are static conventions that are apparent in the context of individual phrases, we will argue that "syntagmatic" models address the underlying dynamics governing the evolution, entrenchment, and cognitive internalization of syntax; notions of information content thereby serve as a hypothesis concerning the principles underlying syntagmatic dynamics. Moreover, we claim that these interlinked notions of information content, syntactic and semantic constructions, and "path" models of the interface between them, provide a foundation for grammars that have applications in programming languages and data modeling as well as (natural) linguistics. In particular, this theory points to connections between natural-language parse-graphs and formal representations of source code (apart from the obvious similarity between two models that both diagram structures derived from applying parsers to an input language): specifically, one can examine graphs in both contexts in terms of *directions* of information increase. We seek to codify the informal picture that graphs have paths along which a measure of information content (in some sense) elevates from site to site.

There are additional details that may be added to this graph-theoretic picture, but the central

determined by syntactic (likewise, or *syntagmatic*) constructions if our cognitive construal of a given paradigmatic construction, a given construction on the paradigmatic "pole" of language, is *caused by* our applying certain *rules* to synthesize syntactic elements into a syntactic construction.

[2] In effect, we can incorporate certain notions from *situational semantics* (*see*, e.g., [11], [39] or [43]) paired with *construction grammar*, with the connections between the syntax and semantics modeled through what we will call (departing from both situation and construction theories) "paths."

[3] Taking seriously Juan Uriagereka's metaphor in [54] that "syntax carves a path" that semantics "follows," though without committing to such following being automatic or possible without interpretive effort (which can be visualized by saying that a "syntagmatic path" can actually engender multiple "paradigmatic" paths, and that nuanced interpretation may be needed to disambiguate them).

point is that graphs meeting certain criteria (which we will briefly define here, deferring to later chapters a linguistic justification for this model) codify intuitions about cognitive and conceptual processes guiding the dynamics (the "syntagmatic" tendencies) that underlie concrete syntax. For the sake of discussion, we will refer to such graphs as "Syntagmatic" graphs, a terminology which is only distantly related to the usual sense of "syntagma," but which captures the notion that these specific graph-structures are stipulated according to desiderata arising from the syntagmatic dynamics operating within language (or so claimed by our underlying theory). In this sense, the discussion below will describe *syntagmatic graphs* as a specific class (or maybe Category) of graphs which are equipped with certain structural rules and features. Syntagmatic graphs differ in some respect from the categories examined in Coecke *et al.*, but arguably these graphs capture many of the details and intuitions informing those authors' use of hypergraph categories, and, in general, syntagmatic graphs can therefore provide one system for representing structures in natural language. However, we will also argue here that syntagmatic graphs may be useful tools for representing *formal* (e.g., programming) languages.

Coecke *et al.*'s reasons for adopting hypergraph categories to model natural-language grammar were not, first and foremost, philosophical; instead, they were looking for representations of language that are conducive to analysis or recapitulation via operations that can be programmed on a Quantum computer. As such, the value of hypergraphs in this context lies above all in how hypergraphs organize data so that operations on data can be "compiled to," or implemented in terms of, a computational machine instantiating quantum-computing logic.

Our proposals here are analogous in that we recommend a specific variety of hypergraphs (syntagmatic graphs), whose properties are selected to facilitate implementation of a virtual machine to evaluate queries over data sets and data models. As (summarily) formulated here, this virtual machine would be classical (not Quantum).[4] Nevertheless, the overarching methodology here applies for both Quantum and classical settings: graph models can be articulated with particular concern for *the nature of virtual machines which evaluate queries against the graphs* and, moreover, problems in multiple domains (e.g., data integration and NLP), which are not apparently graph-theoretic on the surface, can be translated to *queries against graph structures*. In combination, these principles point to criteria for optimally formulating graph metamodels: pay attention to how real-world problem domains give rise to graph-queries, and then to the design of virtual machines evaluating those queries, which are constrained by the structures of graphs encoding the queried information. This chapter will examine these dynamics in the context of (what we call) syntagmatic graphs.

6.2 Syntagmatic graphs and pointcut expression semantics

The last few paragraphs have outlined, at least in brief sketch, what amounts to a "theory of language" in which the crucial dynamic of linguistic understanding—manifest in both syntax and semantics—is a certain expansion in information content driven by an effective coordination between semantic and syntactic constructions. On one level, this may seem obvious (it is hardly newsworthy to claim that sentences convey information) but we intend to emphasize the idea that information is accumulated in

[4] Although via parallelization tactics similar to the Gremlin virtual machine (for property-graph query evaluation), it could conceivably be optimized by factoring certain algorithms into Quantum form (perhaps using Coecke's existing quantum-computing hypergraph models as a starting point).

stages, and that these stages thereby can be modeled via *paths* situated in both semantic and syntactic "spaces" or structures (of some kind still to be elaborated). We can then connect linguistic models (such as Coecke *et al.*'s mixture of hypergraph categories and conceptual spaces) to computational and graph-theoretic contexts wherein notions of *path* and *information content* are well-defined.

A full discussion of this theory in linguistic terms is outside the scope of this chapter, but what *is* relevant here is the structural format that emerges when we try to express this linguistic picture in formal models, e.g., parse-graphs. The meaning of individual linguistic units, in particular, are the details they provide—the information content they contribute—to the accretion of detail vis-à-vis an overarching semantic construction. As we will argue in Chapter 9, grammars fluidly modeling this information-theoretic perspective are generally "verb centric," insofar as semantic constructions are canonically designations of states, events, or processes that are concretized in the root-verb of a relevant clause. Parse-graphs, for example, would embody word-to-word relations where each word adds detail to a verb-profiled scenario, and where "paths" tracing word-to-word connections lead toward verb-nodes, following "contours" of greater information content. In graph-theoretic terms, this implies directed, labeled graphs (the labels being lexemes/morphemes and parts of speech on nodes, and inter-word relation kinds, as in link grammar [50], [12], [51], on edges) with the property that every directed edge belongs to a path leading to a verb-node and verb-nodes are, in general, *targets* (rather than *source-nodes*) of edges. Edges can be classified in terms of the *kind* of information they contribute to their construction (see Fig. 6.1 for a visual diagram of this model).

For the sake of discussion, we will stipulate that verb-nodes are always *only* target-nodes. Of course, verbs form clauses which can be incorporated into other verb-profilings (as their own

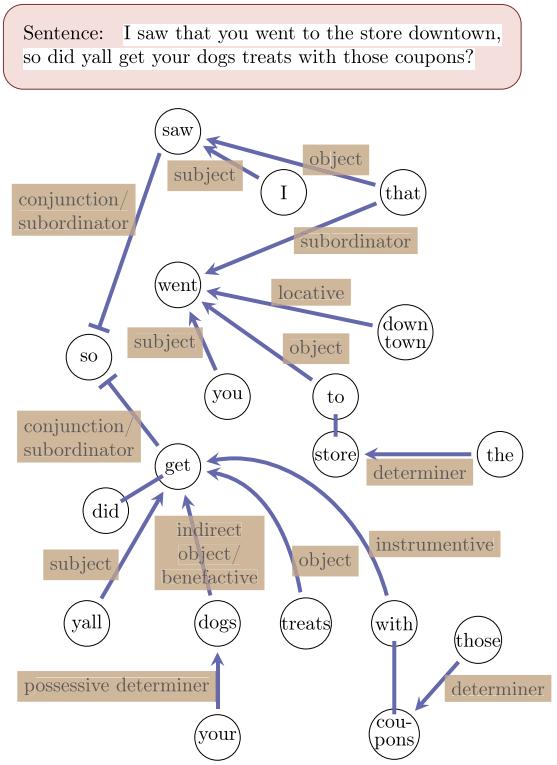


FIGURE 6.1 Semantic constructions as (verb-centered) information-amplification pathways.

details); this is why sentences can have more than one verb. In many cases nested clauses are "packaged" via hinge-words such as "that," as in *I saw that you went to the store*: here *went* has its own cluster of details (its subject, say, is *you*), while *saw* has a different cluster (among other things, *it's* subject is *I*). The pivot that bridges these two different foci—we go from in some sense picturing or learning about the occasion of someone going to a store to learning about the fact that this event was *witnessed* directly or indirectly by the speaker—is the conjunction *that*, which we can say points in two different directions: on the one hand, it connects to the inner clause which it "packages" and, on the other, to the outer clause that draws in the inner clause as a component part (usually playing a noun

role; e.g., a going-to-the-store as object of *saw*). As this suggests, the paths leading *between* verbs can be defined in terms of hinge-nodes that are incident, as source nodes, to two different edges targeting two different verb-nodes. This form of structure may be introduced as a defining principle of Syntagmatic Graphs to preserve the stipulation that verbs are always target nodes rather than source nodes.

Such a convention differs from Coecke *et al.*, who approach verbs from the perspective of category-theoretic morphisms; in graph terms, their perspective implies that verbs have *inputs* and *outputs*, analogous to mathematical functions, which carries over to verb-nodes having *input edges* and *output edges*. However, in practice, programming language runtimes generally hold the results of procedure-calls in "temporary" values, if not lexically or globally scoped variables. As such, unfolding the execution sequence when the result of one procedure is passed to another, there is indeed an intermediary value of some sort—which may be subject to further actions, such as a type-cast or, say, in C++, call to a copy-constructor—and which for graph-representations would be roughly analogous to a linguistic conjunction-node, such as *that*.

We therefore propose a modification of the conventional scheme for encoding syntactic structures in parse-graphs (whether for natural or formal languages). In the model used here, directed edges always point *toward* nodes representing procedures/verbs (the distinction between "inputs" and "outputs" thereby being established by annotations on edges or their source nodes, rather than by edge-direction).

Such an inversion may seem to be a minor notational variation, but it arguably points to a larger shift in perspective, which can be considered in terms of both natural and programming languages. In the natural-language context, we have sketched a theory whose central feature construes semantics in terms of amplification in information content. In Chapter 9, we will more specifically define information content in terms of "accretion of detail" vis-à-vis verbs in particular. This means that for each verb we can identify a collection of "details" associated with the verb, which are all "fed" into some cognitive process that conceives an event/state/process which the verb profiles. All meaning-content in a sentence is therefore "pulled in" to one or another cognitive verb-profiling. At the same time, the full mass of relevant detail is brought to bear on the cognitive process; we are able to cognize the verb-profiling the speaker intends to communicate because we mentally assemble the details relevant to the verb as a precondition for considering the verb (or its associated meanings) as a cognitive phenomenon.

Via the analogy of natural-language verbs to programming-language procedures, we can similarly see a running program as undergoing an *accretion of computational content,* which results in there being sufficient data available to execute a given procedure. Procedure-calls in formal computer science are often considered in terms of "reductions," as in lambda calculus: a procedure call "reduces" to the value that the procedure returns, in the sense that this value takes the logical place of the procedure in subsequent code (sometimes an optimizing compiler will literally replace a procedure's return value for the procedure-call itself if that value can be determined at compile time). However, an equally salient picture is that before the lambda "reduction," there is an *expansion* in the sense that the total data for a procedure-call accumulates by determining (if necessary, evaluating nested expressions) the values for all the procedure's input parameters.

This analogy fits better with process calculi in lieu of lambda calculi, in that (in the language of process-calculi and Petri nets) a procedure can only be called when all "bindings" or "markings" are in place (*see*, e.g., [34] or [35]). As such, a "Syntagmatic" computational model can be defined in terms of computations instantiating evolutionary processes whereby data

is accreted (according to specifications registered by parameter-bindings) and must reach a threshold of completeness for a procedure to be called or "fired." (Still borrowing Petri net terminology: values are "fed into" a procedure, much like coins into a vending machine, one of the examples often used to illustrate Petri nets; procedure outputs are then akin to your candy bars released by the machine when it has enough coins). Once a satisfactory level of information content is reached—by resolving all symbols and nested expressions forming procedural inputs—calling the actual procedure is the next computational step. Indeed, in some distributed-object systems (such as Vission [52]), such an "accretion of content" provides the explicit form of the calling-interface: the origin code constructs values and binds them to "slots" in a remote environment, then (when all needed bindings are defined) explicitly signals that the procedure is ready for execution.

As much as vending-machine-type analogies are popular small examples of Petri nets (or, say, finite state machines), an interesting detail that rarely seems to get noted in this case is how adding coins to the machine does not merely *change the state* of the system. It also *augments the amount* of money in the machine. In other words, we have a natural ordering of most or all possible states based on the total value of all coins deposited at the point when a given state obtains. This ordering affects state-transition rules, because the rationale for transitioning to an *end* state—not just awaiting more coins, but rather releasing a candy bar and resetting to a ground (no coins) state—is that the total value of coins dropped overtakes the target value of the purchased product (we represent this scenario pictorially in Fig. 6.2).

Our analogy between vending machines and procedure-calls derives from the idea that evaluating nested expressions—or reading the value of symbols which are input as single tokens—can be modeled as adding "information content" akin to a total sum of money rising with

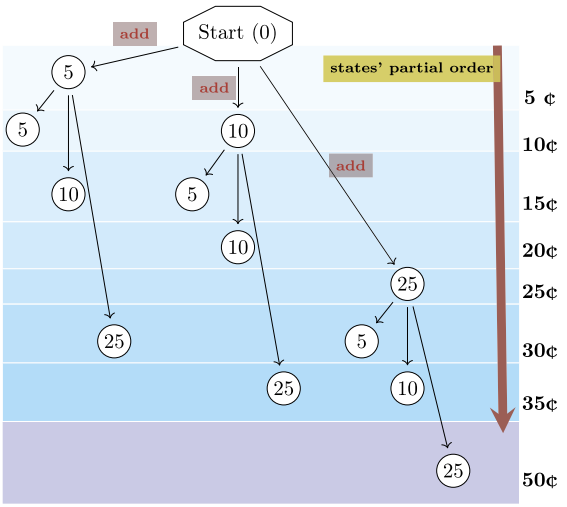


FIGURE 6.2 Diagram of a "vending machine" state chart showing states' partial order (based on total coin value).

each vending-machine coin. The idea of information content, and of ordering systems' states in terms of variant *amounts* of such content, introduces new semantic possibilities to the potential utilization of graph or hypergraph-based representations for different data structures. We think this intuitive notion has possible applications to natural language as well as formal (programming) languages. Indeed, we will sketch out a theory that finds certain overlaps between the syntax and the semantics applicable for several different domains: (1) representations of natural language; (2) representations of programming languages and source code; (3) representations of assertions and beliefs within data and "knowledge" bases; and (4) serializations of data structures.[5] We will point out how introducing *increase in information content* as a

[5]For this most part this book uses natural-language formations as motivating examples (on the premise that formal-language semantics can emulate linguistics proper all else being equal) and we are not focusing on NLP itself, although our discussions in the natural-language context could point to a query-architecture for text-mining and textual database engineering, potentially.

core structural feature within these diverse domains helps to solidify our understanding of their overlap, which could potentially be underdetermined without a theoretical background: though there may be superficial connections between (say) natural and programming languages, or between source code and serialized data structures, we need a theory that can leverage such similarities for them to have any practical value.

In itself, a generic appeal to "information content" does not carry very much theoretical weight either, but it *can* serve as structural ground for more well-developed syntactic or semantic accounts. For example, information content could be leveraged in the context of natural language's distinction between "theta" and "thematic" roles (to be discussed further in Chapter 9); theta-roles derive from verb/subject or verb/object pairings, whereas thematic roles derive from syntactically more distant details (consider *I drove some students to a conference in Philadelphia*: the *conference* and *Philadelphia* are *thematic* in their grammatic relation to the verb, and likewise are more peripheral semantically than the "thetic" verb-subject *I* or object *some students*). This distinction at the level of clause formation primes us to place different levels of emphasis on the information contributed by theta-role content versus thematic-role phrases.

In the context of programming languages, we can identify different *varieties* of information content. Some of these are (potentially) determined as a static invariant within source code, such as the types of every parameter passed in to a procedure, or such as a procedure's memory address (which is thereby associated with any source-code token naming that procedure). Some of these details may *or may not* be fully resolved at compile-time. For example, a virtual method would typically be referenced via a source-code token that maps to multiple procedures at runtime (based on the actual type of its **this** object) and argument types (assuming we distinguish supertypes from subtypes) would likewise be opaque for static analysis. Meanwhile, specifics such as the actual values of procedure arguments would (outside of unusual cases) be open-ended until the runtime procedure is actually called.

We can compare these cases through the lens of information being added on at different times—some details are fixed by the compiler, some deferred to runtime, and some occupy an intermediate status, where both "early binding" or "late binding" is possible. Given such a classification for information-content vis-à-vis procedures, we can then consider how details *other* than type-attributions and value-bindings fit in to the compile-time/runtime contrast, such as argument ranges and typestates in conjunction with procedural preconditions. The net result would then be a model for computer code that theoretically buttresses the abstract idea of *information content* with a classification system structured around notions of compile-time/runtime and early/late binding.

In the same way that natural-language semantics can be glossed with the picture that all meaning provides details which supply information content for a verb, we can adopt the perspective that computational processes involve declared or evaluated values supplying data for a procedure-call: any time there is a concrete value that is part of a program's running state, there is a procedure toward which that value will be targeted as one requisite parameter. To fix the "meaning" of a value, then, we need to identify the procedure where that value is used. This is one rationale for adopting a convention wherein all nodes not themselves representing procedures (instead encoding literals or scoped symbols) point toward procedure-nodes. In such a graph, each procedure-node is surrounded by nodes representing data that must be concretized (such as input values that will be bound to the procedure's input parameters) as a precondition for the procedure to be runnable. Although when looking at source code we may see this as a static phenomenon (a single expres-

sion being a unit of a computation, analogous to one clause being a unit of linguistic meaning), during execution this process unfolds over multiple stages, wherein each step represents one piece of preconditional data being set in place anticipating a procedure call.

Such a picture justifies the model that *all* non-procedure nodes, even those representing output values, become source-nodes of edges directed *toward* procedures. For example, a procedure can only be called if memory is allocated to hold the procedure's return value—although the actual return value is not provided until the procedure itself runs its course (being as such a *postcondition* of the procedure), it is a *precondition* that the return value will be "held" somewhere, so the edge connecting the node representing this "carrier" (using terms from Chapter 5) can point *toward* the procedure node, because the carrier (albeit not an actual value) is a precondition for the procedure running. Output and input nodes are therefore *both* procedure-preconditions; the difference is that input-nodes are preconditional in needing an actual *value*, whereas output nodes need only represent the capacity to *hold* a value.[6]

In the case of natural language parse-graphs, "bridge" nodes between verbs represent parts of speech such as subordinators/conjunctions (e.g., *that*). The analogous construction in programming languages would be output nodes for one procedure becoming input nodes for a "parent" procedure (vis-à-vis expression nesting). However, the model of syntagmatic graphs presented here proposes a modification of this scheme, in which such "pivot" nodes are essentially split in two. In general, we want each node in a parse-graph to be *uniquely* adjacent to one procedure-node (nodes are uniquely adjacent if they are adjacent to only one edge whose other node has the relevant unique property, in this case that of being a procedure node). Instead of a single pivot node therefore, with two different procedure target-nodes, we prefer a pivot "edge" connecting two distinct nodes, one representing the output of a procedure and the other the input of a different (parent-expression) procedure, with the *edge* between those non-procedure nodes representing the phenomenon of one's return value being the other's input parameter (the edge thereby representing source-code sites and program-runtime stages where details such as type-cases and temporary-object constructors/destructors come into play). We present this as a further stipulation on syntagmatic graphs intended to model computer code.

This overview does not fully exhaust all considerations needed to circumscribe the relevant class or Category of syntagmatic graphs. One point we have not yet mentioned is that the actual procedures called in a program might itself

[6] As an example, the **[[nodiscard]]** attribute in C++ enforces a requirement that the result of a procedure be accounted for—bound to a variable or passed to another procedure—which means that the context for the call to proceed is not correct without additional code that uses that return value. This situation is roughly analogous to passing an argument by reference which is intended to hold the result of a computation (a common pattern in languages that can only return one sole value from functions; if multiple returns are needed, the others are handled instead by overwritable references, which the called procedure initializes so as to communicate the value that would otherwise be returned). A non-optional parameter must, of course, be passed in to a procedure for the call to be possible. But passed-in parameters sometimes serve the *semantic* role of being initialized (or re-initialized) as if they were return values; in short, the syntactic difference between a return value (as bound to, say, *y* in **y = f(x)**) and an overwritten reference parameter is a surface-level distinction; semantically the two constructions are similar. Insofar as each (non-optional) parameter is a *precondition* for the procedure-call, then there is no reason not to consider a *return* value (or more precisely some mechanism in the calling context to use that value) as equally a precondition (which is appropriately modeled by connecting a node representing this value *to* the procedure node, rather than having the edge point in the opposite direction). After all, both a (non-optional) parameter and a (non-discarded) use of a return value are preconditions for a procedure-call (in this analogy, the *absence* of a non-discard attribute would be equivalent to allowing an *input* parameter to be optional, rather than required).

be evaluated at runtime, so that what appears as a procedure node in source code may in fact be a procedure, which *yields* a procedure which *then* gets called; in other words, a single procedure node in this kind of case embodies *two* (or more) procedure-calls, where the first yields a value that holds the second (as a function-pointer, say), the actual procedure called to complete the expression not being known until runtime (until, that is, the first procedure returns, and then in turn other layers' if applicable). This scenario can be accommodated by allowing inputs/outputs to a procedure node to be sorted into layers, one layer representing an expression which when called yields a procedural *value*, which in turn constitutes the procedure whose parameters are bound through the second layer, and (potentially) so on iteratively. The analogous construction in natural language would be adverbs modifying verbs to form verb-phrases; it is these modified "verbs" that form the nexus of "details" informing the relevant verb-profiling.

With these further constructions at least sketched provisionally, we can clarify the specific properties of syntagmatic graphs that for this analysis we employ to model programming language code. Syntagmatic graphs for these purposes are characterized by properties such as the following:

Procedure nodes Nodes in general are labeled with type-attributions, and by extension nodes whose types represent procedural values (those with input and/or output parameters and/or side effects, as opposed to static values) can be distinguished from other nodes;

Procedure nodes as target nodes Procedure nodes are *target nodes*, but never *source nodes* for edges labeled with annotations representing computational constructions, i.e., the label semantics requisite to render directed labeled graphs as models of computational processes, embodying lambda or process calculi (this does not preclude procedure-nodes being potential source nodes from a fully distinct class of edge-labels, such as those representing sequencing between distinct source-code statements);

Procedure nodes as path targets All paths (when constructed via edges whose label semantics represent the subset of labels specific to modeling computational processes, as in the previous item) terminate in procedure nodes, and each procedure node is adjacent to a collection of non-procedure nodes, each of which are adjacent to no other procedure node—for the sake of discussion, although this is technically a non-standard way of using the term, the set of a procedure node and its "surrounding" input/output nodes can be called a "neighborhood," so that a program's source-code graph can be decomposed into distinct neighborhoods, each representing a distinct expression[7];

Carrier-transfers as inter-neighborhood pivots With source-code graphs divided into neighborhoods in this manner, a specific group of edge-kinds can be defined to represent bridges between expressions, wherein the node representing one procedure's return value is connected (in the sense of passing a value to) one representing another procedure's input (expressing "carrier-transfer" semantics, as described in Chapter 5);

Channels and layers Edges incident to a common target-node can be grouped into "channels," as defined in Chapter 5, representing collections of input or output nodes with similar semantics (e.g., regular inputs are a channel distinct from object message-receivers, which have a different semantics, and regular outputs are distinct from thrown exceptions), and "layers" to accommodate where procedures return values that are themselves other procedures; which are then called with bindings

[7]To be precise, we could define a *neighborhood* in a *directed* graph (this is a term more common in undirected graph theory) as the in-neighborhood of a node with outdegree zero.

derived from a different collection of nodes in the same "neighborhood."[8]

Syntagmatic graphs meeting these criteria are *hypergraphs,* because edges incident to a given procedure-node are grouped together; instead of one input and one output node, we have edge-collections structured via channels and layers. Coecke *et al.* generalize graphs to hypergraph categories so as to model similar arity-variance among inputs and/or outputs (procedures may input and/or output zero, one, or more than one value), though (as outlined in the prior paragraphs) syntagmatic graphs have additional structural details than defined for those authors' hypergraph categories. We contend that syntagmatic graphs, as defined here, serve similar roles to their hypergraphs, but being somewhat more expressive, present more detailed modeling features.

6.2.1 Query-evaluation foundations for syntagmatic graphs

Our rationale in setting out schemata such as these definitions/constraints for syntagmatic graphs is not primarily presentational. As we have intimated, there are numerous non-isomorphic or not-fully compatible but nonetheless similar structures that can model computational phenomena, such as procedure-calls; structural differences may iconify philosophical variations in how we choose to "think about" computation (or, say, formal data models), but it should not be taken for granted that subtly alternative philosophical framings are consequential from the perspective of implementing useful technologies. For our purposes, the importance of structures such as syntagmatic graphs lies in the project of codifying the operational and implementational environments contextualizing technologies that work on data structures, such as graphs as digital resources. This involves graph query languages and the engines that evaluate such queries, as well as software for representing and serializing graphs, which in turn should be designed with consideration to how query-evaluation engines would work; the in-memory and on-disk representations of graphs provide the structures on which graph query engines operate.

Structural features defined for a particular kind of graph correspond to graph-elements that may be identified within queries, incorporated into underlying representations, and accounted for when evaluating queries against such representations. Structural features can be motivated, as such, based on whether they make queries more expressive and/or whether they support representational devices that are helpful for query-evaluation. In this sense distinct structural formats (e.g., several different graph Categories) may have noticeable differences in terms of the forms of queries they engender, and the ease and flexibility with which those queries are evaluated and adapted to problem-domains based on how a graph technology is being used, in context.

For example, consider the edge marked "carrier transfer" in Fig. 6.3 (between nodes **<r>** and **<l>**). This edge does not represent an example

[8]To expand upon the notion of "neighborhood" introduced in Footnote 7, we'll call a neighborhood as defined there a "0-neighborhood," meaning the in-neighborhood of an outdegree-0 node (called the *center* node), and a "1-neighborhood" to be the in-neighborhood of an outdegree-1 node. A "1-neighborhood" therefore has one edge, which can be called the "escape" edge, referring to the one edge incident to the neighborhood center node as *source* (or *(direct) predecessor,* for the terms *predecessor/successor* being common expressions in lieu of what we call *source* and *target*). By analogy, we can say that an *escape edge* is an edge which is source-incident to a node in a 0-neighborhood, which is *not* the center node; a 0-neighborhood would have a *unique* escape-edge if there is only one escape edge (e.g., when modeling a procedure call in an environment where procedures may not return multiple values). An *escape channel* would be a channel formed by aggregating all escape-edges in a 0-neighborhood. A *layered neighborhood* can then be a neighborhood where non-center nodes are indexed by a number denoting a layer; we can stipulate that an escape node/channel on a neighborhood may only include nodes from the highest-index layer.

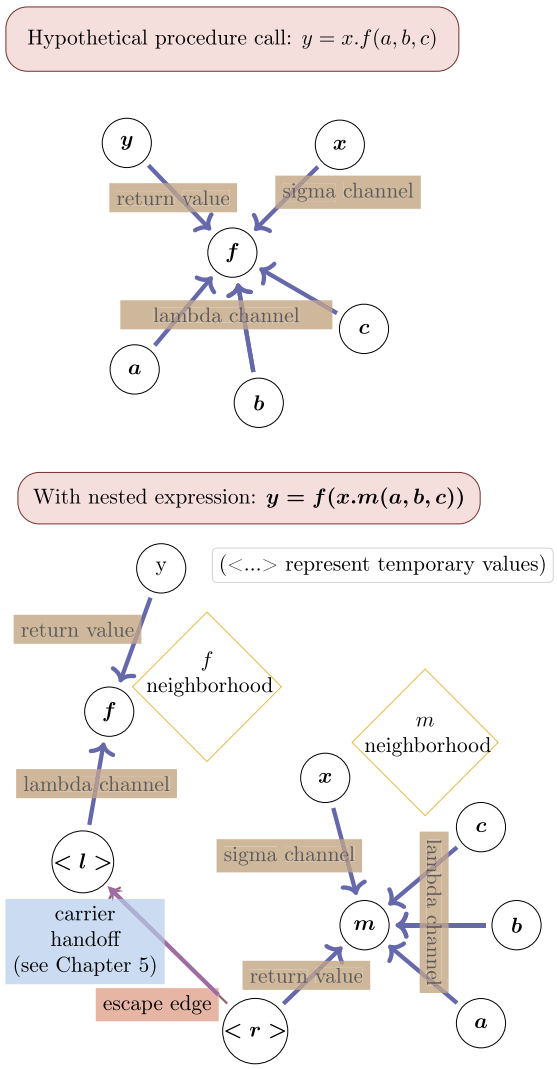


FIGURE 6.3 Neighborhoods in syntagmatic graphs.

of the kind of program execution step (i.e., a "join-point") that is apparent on the procedural (source-code) level, but it could be defined as analogous to a join-point on a "subprocedural scale" (we'll discuss these terms below). Moreover, this edge describes a consequential point in program execution, because it is possible that operations will be performed behind-the-scenes here (e.g., type casts and class constructors/destructors). It would be beneficial, accordingly, to implement a pointcut-expression system wherein designating that specific runtime moment is possible, motivating the use of multiple nodes with "escape" edges and handoffs to represent return values being passed to outer procedures.

As a terminological aside: we employ the phrase "hypergraph queries" to encompass several kinds of expressions that could be evaluated in the context of hypergraph structures and databases (following conventional usage in the context of formal languages for manipulating databases). More specific kinds of hypergraph queries would include *traversals* (logic for moving between different graph-locations); *descriptions* (stipulating conditions on hypergraphs for a specific purpose); *site-locating*, or more concisely *siting*, referring to finding graph-locations meeting some criterion; *serialization* (encoding a hypergraph so that it may be shared between different applications or computing environments), and *extraction* (pulling information out a hypergraph, by matching the structure around some *site* with a *shape* specifying how this structure encodes data that is expressed in that part of the graph).[9]

[9]In graph databases in general, a *shape constraint* designates requirements on the structures evident between graph elements — e.g., which edges exist between nodes taking into consideration edge and node labels — which then indirectly encodes how the graph structure is used to express data for that specific graph-encoding scenario. The canonical example of a shape-constraint would be using a collection of nodes to encode named data fields of a central data value: one node represents that value, and it is connected via labeled edges whose labels reproduce (or at least are uniquely associated with) data-field names, where the incident nodes of each edge are labeled with the corresponding field values (a **last-name** label annotates the edge pointing to a node whose label is a person's last name, for example). A shape constraint thus stipulates that a given node must have at least those edges whose labels correspond to fields which must be present for the value represented by the central node to be correctly instantiated. Conversely, then, extracting data

Our general goal in this book is to reduce data-integration problems to hypergraph query problems. In other words, assuming we have a hypergraph-representation strategy that can support the various additional constructions we have been or will be setting forth (channels, layers, neighborhoods, and so on), such a paradigm lays the foundation for a collection of primitive operations on hypergraphs (thus defined) and the possibility of decomposing hypergraph-query algorithms to a set of such operations. Our claim that many data-integration problems can be reduced to hypergraph queries does not entail that a query language designed for data-integration solutions should explicitly refer to "low-level" hypergraph terms or concepts; instead, the high-level query language should employ vocabulary and grammatical formations that reflect the high-level domain relevant to each part of the language (e.g., pointcuts and pre-persistence representations, using terms to be discussed later in this chapter).

That being said, insofar as query evaluations on these various domains are "reducible" to hypergraph queries, then it should be possible to (within the requisite query-processing engine) translate or "compile" the high-level concepts to hypergraph concepts that can be addressed by the hypergraph operations just mentioned. These principles encapsulate our query-evaluation strategy; although this chapter presents only a summary insofar as we do not identify examples of these kernel operations or translations explicitly, this overview sketches out what would constitute a complete theory of the relevant hypergraph algorithms and the compiler-architecture to translate high-level queries to those algorithms.

from such graph-sites — e.g., getting a person's last name — leverages the appropriate shape-constraint to determine which graph site (e.g., which node) holds or can be manipulated to obtain the desired information (shape-constraints are analyzed in e.g. [22], [42], [18]).

6.2.2 Use-cases for source-code graphs

Our "syntagmatic" graph model has more structural detail than other systems for representing computer code—such as abstract syntax trees or Semantic Web style RDF graphs—and one claim of this chapter is that this distinct form of graphs bears consequential differences to other code-representation strategies, rather than being just a superficially different notational convention. We need to explain, however, why the representations proposed here go beyond mere notational variations (manifest mostly at a meta-discursive level); in other words, why these variations in how source-code is *described* translate to variations in algorithms or logics for how source code is analyzed or otherwise manipulated.

To address this question, note that there are several different reasons why formal representations of computer code are important. One reason is that such representations are a precondition for translating high-level source code to machine or virtual-machine instructions, but even outside of compiler theory, there are several use-cases for formal code representations. One is the fact that source-code files represent digital assets in their own right, which are interactive/visual objects in the context of software programs such as integrated development environments (IDEs), where users expect features such as syntax highlighting and code completion—displaying code with different colors for tokens bearing different roles (variables distinct from procedure-names and from literals, for example), and with interactive capabilities such as providing information about procedures via context-menus bound to source-code tokens bearing their names. Displaying source-code files as interactive documents in this manner is only possible when IDEs can build structural models of source code, rather than treating the files as unstructured text files.

Another use-case is run-time reflection: examining source-code allows programs to make

certain capabilities (such as executing a procedure by supplying its name dynamically) available at runtime (more on this kind of use-case below); or aspect-oriented behavior modification (where code is added to adjust how the program behaves at certain specific execution-points). Moreover, an additional scenario—which overlaps with aspect-orientation in this sense—is that source files need to be processed as "queryable" assets, where we can search for one (or multiple) code-locations meeting some specific criterion (an example would be applying aspect-oriented modifications at the locations thereby identified, but such code-searching equally well applies to using IDEs, or to scripts composed for code-analysis, where for instance programmers conduct code-review by locating points in code that reflect specific design patterns, or which tend to be sources of bugs and coding errors).

In short, a number of different kinds of software components—including compilers, IDEs, and code-libraries, which implement various runtime-reflection and/or aspect-oriented runtime modification systems—need to construct and manipulate formal representations of computer code. Therefore, strategies for code-representation are important insofar as precise and expressive representations make such components easier to design and implement. Formal source-code models come into play at several stages of working with code-representations, including building such representations in the first place; although code-models can be instantiated by parsing source code directly, in many contexts one can construct more refined code-representations by extending a given code base with annotations, either inline with or external to the code itself. Therefore one use-case for code models is designing languages for *annotating* the code that is actually seen by a compiler.

A second use-case is designing languages for *describing* code-locations subject to further examination or processing; in aspect-oriented parlance, such location-descriptions are called "pointcut expressions." So an essential dimension of code-annotation languages is the capability to formulate pointcut expressions, which can be used to select points of interest (often called *join points*) within a body of source code.[10] A further use-case for code representation is to actually support such designations; that is, to traverse source code to specifically identify the pointcuts described by a given pointcut expression. Code representations can be more or less efficacious in terms of how well they support such query-guided traversals to evaluate pointcut expressions.

Note that how best to articulate the semantics of pointcut expressions is an open question; there are several options for making explicit the general idea that join-points represent "locations" in source code. That could literally mean points in source code taken as a textual document, but it can also mean some step in the runtime execution of a program, which in the latter case engenders a need to define "execution steps." Ultimately, the individual operations that collectively constitute a running computer program are determined by the computer's machine language, but it is not straightforward to map source-code-level join points to individual machine-language instructions. Amongst other problems, the same source code may compile to different kinds of machine language for different operating systems (and will also vary according to which compiler is being used). So, if machine language instruction-sets were chosen to provide rigorous semantics for join-points and pointcuts, these semantics would have to be defined separately for each operating system and compiler.

An alternative is to consider abstract or *virtual* machines designed, at least in part, to provide this kind of semantic framework for defining program execution steps (at a finer scale than high-level source code). Another alternative is

[10]Hence a *pointcut* is a set of *join points*, and *pointcut expression* is a formula for filtering join points into a pointcut.

to define source-code locations not in terms of actual source code, but in terms of that code as compiled to some graph- or tree-like representation (such as syntagmatic graphs). Our approach is compatible with either of these final two options, or indeed with a combination of the two.

Moreover, as we will discuss later in this chapter, pointcut-semantics can extend over different "scales" in source code (or program execution). Consider, for example, a remote procedure-call (RPC), which is carried out by encoding a message with instructions to perform some calculation exposed as a web service on a remote machine, with the request itself encoded in HTTP. This HTTP content is not itself a procedure-call, and would not adhere to the conventions of an ABI (but rather to the remote service API: application *programming* interface, rather than *binary* interface), but in terms of code design and purpose such a web request can play a *role* akin to a local function-call. Also, remote and local procedure-calls have enough structural similarity (in terms of their pre- and post-conditions) that one can certainly use the same sorts of expressions to describe calls of both kinds. Given these considerations, it is reasonable that a pointcut expression language could be designed to generalize to broader phenomena, such as remote procedure-calls, which (we propose below) involves a notion of multiple Syntagmatic "scales." In this guise pointcut expression languages would overlap with service-description (SDL) languages and with analyses to confirm that an RPC conforms to the target SDL protocol.

The practical consequences of models such as Coecke *et al.*'s hypergraph categories—and, we propose, variations such as syntagmatic graphs as in this chapter—are such that these various lines of research can point to more effective code-representation strategies, which can then lead (among other results) to more detailed or powerful pointcut-expression languages and evaluators. The open question of pointcut expression semantics in aspect-oriented programming, as well as in software language engineering in general—we should note that the notion of pointcut expressions is not limited to Aspect-Oriented paradigms exclusively, but also applies to debugging, static analysis, and runtime-reflection scenarios (*see* [6] and [28] for a good overview)—cuts across both source-code-representation theories and compiler/virtual-machine design. It is obvious that pointcuts are collections of "locations" in source code, but such a definition needs to be more rigorous, because source-code is a structured system (not just text) and locations represent sites within those structures. Applying models such as hypergraph categories or syntagmatic graphs allows us to ground these semantics by defining source-code structures in hypergraph terms. In effect, pointcut expressions thereby become a genre of *hypergraph queries*—designations of hypergraph sites—so that pointcut-expression languages may be treated (and implemented) as subsets of more general hypergraph-query languages. Syntagmatic graphs, as a form of hypergraphs formulated to represent computer code, can thereby provide a semantic structure-space suitable for a general-purpose pointcut-expression semantics.

This discussion has therefore presented one specific rationale for pursuing syntagmatic graphs as a theoretical construct; we claim that such graphs form an effective framework for designing and implementing pointcut-expression languages, and by extension for code-modeling in general. However, our overarching goal is not code-modeling *per se*, but *data* modeling; so we will also clarify how code and data modeling can be interconnected.

6.3 Applying pointcut expressions for data modeling

Constructions designed for code models do not automatically translate to *data* models; how-

ever, code and data models often overlap, because one of the specific purposes of computer code is to instantiate data models. This is particularly true if one adopts strongly typed data-modeling conventions, where information spaces in general are decomposed into distinct data types. Almost all data *sets*, for example, can be structured as collections (which may be ordered or unordered sets of individual values/records/objects) whose elements are data structures each having the same type, and accordingly having similar patterns of organization (e.g., the same set of individual data-fields). Dividing a data set into distinct collections-types and individual-value types yields the possibility of modeling some or all of these collections and/or "individual" values (which in general are not *one* single value, but rather single data structures with some record-like structure, e.g., tuples of named attributes) via corresponding data types implemented in computer code.

Even though one could manipulate data sets and/or databases without such code-instantiations in some case (for instance via database queries), it is generally possible *at least in theory* to consider any data type internal to a data set and/or database to be convertible to data types in a programming language, with a corresponding implementation in terms of procedures such as constructors and field-accessors (some of the various procedure-roles were outlined in Chapter 5). This translatability is directly applicable to data integration, because any integration scenario too complex to be achieved via database queries alone may be resolved by providing full-fledged programming-language implementations of any data types that need to contribute values to an integrated data model.

A complicating factor here is that different database models exhibit different levels of schematic rigor: in a JSON-based model where database objects may be arbitrary associative arrays, there is no guarantee that the key-names for an arbitrary object will match field-names for a recognized data type, so strong typing in such an environment is more difficult. However, data models can impose greater structure than required by a database engine itself; marshaling existing data to conform to a more rigorous model can then be one part of a data-integration pipeline. Whether or not concrete implementations are actually used for a given data type in a given project—and considering that integration workflows can include elevating more weakly typed to more strongly typed transpositions of data sources as necessary[11]—we can develop theories related to database management built around strong-typing assumptions with respect to partitioning database or data sets' contents via type-attributions, and expression of data-model constraints in type-theoretic terms.

In practice, a central tool in the data-integration arsenal is that of translating data types internal to a data set/database into implemented data types in some general-purpose programming language. Once that occurs—and in the scope of data-integration scenarios where such translation is useful—we therefore have a body of computer code whose explicit purpose is to provide computational resources manipulating data types that embody information present in a given data source (e.g., a data set or database). It is in this context that specifications in the *data model*, endemic to the data source in question, can be carried over to specifications in the *code model*, manifest in coding constructs and annota-

[11] This may sound as if we are passing off as almost trivial what is often the most difficult part of data integration (a trap lucidly and amusingly captured, in the Semantic Web context, by Clay Shirky: https://www.karmak.org/archive/2004/06/semantic_syllogism.html). However, transposing a data space to a canonical format is not any more complex a problem than integrating multiple strongly typed spaces; so any computational environment wherein the latter is feasible should make the former feasible as well. When weakly typed data sources are an integration hindrance, this suggests a two-step strategy wherein the sources are first normalized as strongly typed assets to begin with, so that type-theoretic constructions are available to strategies at the second (integration) stage.

tions pertaining to the procedures whereby units of data present in the original data source are manifest as typed values in the corresponding computer code. There are many ways in which data and code models in such contexts may overlap.

Strong typing also offers a productive theoretical perspective in which to discuss design patterns for data sets and data-integration strategies. Insofar as data sets are (in principle) strongly classified into types, a canonical (though not exclusive) mode of interacting with data sets is to (via queries of some sort) obtain typed values meeting certain criteria. That is, many queries against a data set will yield responses that take the form of a type-instance, or a collection of type-instances, or an iterator to step through such a collection. Type-instantiation thereby becomes an essential facet of the data set's interactions and query capabilities. In this sense, a crucial detail that we may consider at any site in a data-model instance (which we assume may be logically represented in graph/hypergraph fashion) is *what typed value can be initialized with elements in the neighborhood of this site*? Shape-constraints provide a way to explicitly annotate graph-neighborhoods with type-initialization routines when appropriate. Let's call this the *shape-constrained initialization* (or *instantiation*) concern ("concern" in the sense of a core implementation detail for query engines).

Of course, in the typical scenario we do not happen to be at a site *a priori* and seek to populate a type-instance accordingly; instead, we want to find sites which allow instances to be constructed based on some criterion. In other words, we want to know whether a type-instance initialized from the site will have some property (some data field equaling some value, or lying in some range, etc.). Call this the *type-instance selection* problem. For a given site, we want to know both whether a type-instance can be constructed and, if so, whether it would be selected when filtered through criteria within current query or traversal specification. Note that query optimization should assume that the *selection* problem does not depend on actually constructing type-instances (otherwise we are not really providing query evaluation, only deserialization capabilities and deferring to application code the ability to select type-instances based on logical criteria). That is to say, a database and/or data set should be structured with enough information to resolve selection-problem questions without *completing* type-instantiation, though it should be determined whether instantiation is *possible*.[12]

Note also that the necessary expressiveness of shape-constraints applied to selection/instantiation concerns expands according to the range of graph structures one seeks to target via a query engine. For a hybrid property/hypergraph model, for example, the process of initializing data points can be connected to type-instances through multiple pathways, including property-assertions, hypernode/subvalue relations, inter-hypernode edges, and foreign keys or similar "proxy" values that (under semantic interpretation) connect nodes to other nodes (which may not be explicitly connected via graph mechanisms). Each of these configurations have to be accounted for when providing kernel operations applicable to selection/initialization evaluations, and also when specifying (and parsing) the query-language syntax (see Fig. 6.4 for a visual illustration of these differ-

[12]In some cases, one could then use data leveraged for selection-queries to yield query-results that do not depend on type-instances; this would be in keeping with conventional query paradigms such as SQL. In other words, we can return results akin to what a type instance *would* give for a data-field or some other computation if the instance were fully constructed from the relevant neighborhood. However, in query environments such as those we discuss here, we prefer to consider these "as if" constructions to be merely a special case of querying based on type-instantiation in conjunction with selective filters, where the step of constructing a full instance can be skipped in some cases, but is still logically central to designing and implementing the query engine.

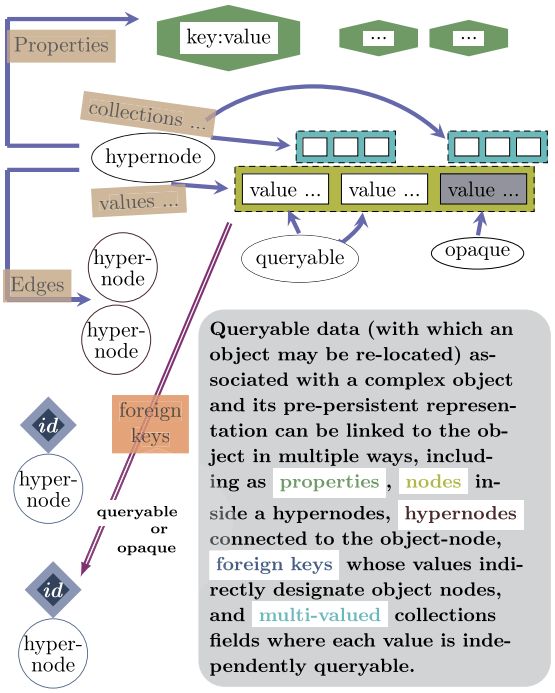


FIGURE 6.4 Query-aware prepersistent representation.

ent modalities, where information pertaining to a type-instance may be accumulated).

We claim in general that selection and type-instantiation problems are the most important desiderata for designing data models and the data sets that instantiate them. In the current context, note also that these are essential considerations for designing query-evaluation engines. At the core of any query-engine virtual machine should be logic for initializing type-instances from a graph-site and for testing hypothetical such instances against selective filters.[13]

[13]In addition to query-processing, we'll also note that VM-based coding environments can simplify certain details in realizing effective (responsive) User Interface designs, as will be demonstrated in this book's demo code. VM methodology can be particularly useful in contexts where multiple sequences of parallel execution can complicate the proper ordering and interconnection between callback procedures. The demo code presents several case studies of this phenomenon in contexts such as local client code communicating with cloud services (via multi-step stateful protocols, e.g. where access tokens and other preparatory conditions are prerequisite for subsequent workflow steps), and also in the context of GUI "wizards" which are designed to allow users to execute a sequences of logically related steps (where behind-the-scenes processing between steps may require the wizard code to synchorize the view-state with the relevant component's "business" logic).

We postpone until Chapter 9 a more thorough discussion of instantiation and selection in these senses; for the time being, note merely that data-model features that render selection and instantiation problems tractable tend to propagate to code-model features in the context of code libraries that implement the corresponding data types. Thus the structural properties of data models that are exercised in selection/instantiation query processing are also relevant to code-models, and tend to be visible in the numerous concerns where code and data models overlap. Some of these will be discussed in greater detail in this chapter.

6.3.1 Code annotation with units of measurement

As a concrete example of data/code overlap, consider the problem of specifying units of measurement (this chapter will address dimensional annotation in terms of code models; we will revisit this issue in later chapters in the context of data models, particularly those connected to image-annotations and image feature-vectors). Units-annotations document the scales of measurement applied to data-points, which are therefore endowed with more structure than just single numbers. For example, in many applications involving computer graphics and digital documents (such as PDFs), *lengths* (on-screen or on-page distances) can be expressed in several different units, including inches, centimeters, millimeters, and "points" (which are $^1/_{72}$nd of an inch), as well as numerous other scales

(in the context of typesetting, and by extension document-viewing software, one considers also lengths determined by font-face designs, such as those based on the letters *m* and *x*, but such scale-units can be set aside for the current discussion, other than to note that a single dimension, such as length, may indeed give rise to a surprisingly large range of potential measurement scales).

One important feature of robust code is ensuring that procedures that operate on lengths (and other unit-denominated measures) check that input parameters match the procedure's requirements (one should not attempt to add inches and centimeters, for example). There are various strategies to enforce unit requirements. One technique is to simply recognize values measured according to different scales simply as different types. Then a procedure that expects measures in (say) points simply cannot be called with lengths in (say) inches, because such a call would not type-check (for the same reason that a procedure expecting integers could not be called with, e.g., a character-string). This forces calling code to explicitly check that local values are defined in units compatible to called procedures' expectations (converting between scales if necessary), which prevents scale-mismatch errors. (One limitation of this approach is that procedures playing the same role often then have to be defined multiple times, for various plausible measurement scales, which is facilitated by template programming but nonetheless leads to a proliferation of distinct procedure implementations, particularly if one tries to be strict about dimensional analysis—for example, the product of two-inch lengths would, scientifically at least, be an *area*, inches *squared*, and in general every multiplication-operation yields a return type different from the input types, so that there is a logically unbounded number of distinct types needed to represent arithmetic operations on a dimensionally typechecked measurement-units system).

An alternative convention is to use a *single* type for a given *dimension* (regardless of measurement scale), but include in the type-instance a data point indicating which scale applies to the associated value. In other words, each instance has *both* a raw (scalar) value and a code indicating the scale applicable to that value (e.g., inches, points, millimeters, or centimeters). Unit-checking (and inter-scale conversions) can therefore be performed at runtime, which is more flexible (even if less foolproof then compile-type checks).

A different approach to scale-measure consistency is to simply define procedures as taking raw values (presuming that they are expressed as or converted to a given scale prior to the call), but ensuring through code analysis, annotation, and/or documentation that procedures will never be called with wrongly scaled value. For instance, if a procedure requires that its input values be scaled in points, we can attempt to verify—by examining every point in source code where this procedure is called—that its values are guaranteed to be measured in points (rather than inches, say). In the code accompanying this book, for example, the procedure that is run when locating a sentence's extrema PDF coordinates receives a "baseline skip" value in (fractional) points, but (in this code base) that procedure is only invoked in the context of having parsed a metadata file, where the relevant baseline-skip measure is deserialized. The deserialization code explicitly checks that the relevant numeric value is properly described (i.e., marked with "pt" according to LaTeX convention as indicator that points are employed as the length-scale); in other words, the deserializer only accepts strings of the form "number-plus-units," where the unit declaration is restricted to "pt." This example demonstrates how scale-consistency may be enforced by simply exercising proper care at any point where unit-denominated values are obtained, and then passed to procedures where scale-measures impinge on the procedure's outcomes.

This kind of explicit code-verification can be confirmed by keeping track of code-locations where such unit-denominated values are read. Although some values are expressed directly in source code, the more common situation is that runtime values derive from one of three sources: database query results; deserializing markup that encapsulates values via binary codes and/or character strings; or GUI components, where values are interactively entered by users.[14] In the GUI case, it is the component's responsibility to declare which units of measurement are assumed when translating the user-visible cues (e.g., a dial or slider's angular or orthogonal position) to a scaled measure (does a virtual thermometer model temperature in Celcius or Farenheit?). In the case of database queries, the database itself should specify units for non-scalar values. In the case of serialization, scale-units may be explicitly marked (by character strings such as "pt" adjoined to numbers or via separate notations, such as XML attributes) or else globally declared for a given markup format (e.g., via an XML DTD). In any event, the procedure that procures a value from a database, markup serialization, or GUI object can verify that it scales this value appropriately by either deferring to the origin-point's documentation of its own units convention, if that is provided, or else by performing explicit checks when those are documented as necessary (e.g., number-plus-unit strings in markup).

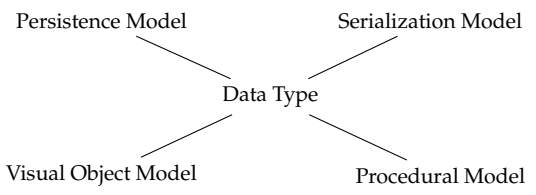
To guarantee that code in this scenario is managing scale integrity properly, it is necessary to identify all procedures that input and output scale-delimited values, and confirm that (1) those which *input* such values specify the unit or units they can properly work on; (2) those which *output* such values either correctly assume that the values are properly scaled within the origin-source from which these values are obtained or else explicitly check for the value's declared units, and perform scale-conversions as necessary; and (3) that every procedure in the first case (inputting scale-delimited values) is paired with a procedure in the second case (outputting scale-delimited values). That is, every time a value is passed as scale-sensitive *input,* we must be able to locate where it originates as scale-verified *output*. Any of these checks are preconditioned on capabilities to identify source-code locations—that is, they all require pointcut expressions, or similar code-site designations.

The example of scale-delimited values therefore presents a use-case of how data-modeling stipulations translate to code-modeling implementation-patterns. The units of measure by which particular data values or fields are expressed is a non-trivial detail of a *data* model. One way that such details become manifest as coding *requirements* is that the code which implements data types instantiating a data model includes sites where the corresponding unit-marked values are obtained and/or used, and these sites collectively represent source-code locations where specifications in the associated data-model would be accommodated and/or verified (assuming the code is implemented properly). In these scenarios, a pointcut expression language (querying for sites in *code*) becomes indirectly a modeling device for the underlying data, because the concretization of the data model in code-implementations serves to document, codify, and exhibit the requirements expressed by the data model. In short, pointcut expressions query (and their evaluators traverse) the code that *documents by implementation* (in the sense of a "reference implementation") specifications pertaining to the relevant data model. For this reason, pointcut expression languages can be considered intrinsic parts of languages for querying (or specifying/validating) *data* models, not just code models.

[14] This does not mean users actually *type* values; instead they may select values from a list, or by manipulating some GUI gadget; consider selecting the temperature on a virtual thermometer by turning a dial or pulling a slider "widget" where, say, clockwise or *up* signifies "warmer."

6.3.2 Documentation by implementation

Insofar as code models *document by instantiating* an underlying data model, it is important to trace the features and aspects of a code body which are directly related to that data model. In general, the relevant code will be any components where associated data-instances are read from (or sent to) a data source, are manipulated and calculated upon, and are (by themselves or as transformed via calculations) shown (statically or interactively) to human users. A particular data-type may therefore be subject to several different transformations or adaptations to specific use-cases, in particular (1) persistence to a database; (2) sharing via serializations; (3) presentation via GUI components; and (4) exposed to different algorithms that extract information from the data (via statistics or any other analyses, depending on the kind of data involved). We can picture this as a diagram where a central data type is figured into four different contexts (database, serialization, GUI, and procedural capabilities implemented to support the analyses relevant to that form of data)—for the sake of discussion, we'll call this sort of diagram a "Semiotic Saltire" ("semiotic" because the transforms of a given data type according to these various adaption-contexts is an example of what Joseph Goguen calls a "semiotic morphism" [19], and "saltire" because this graphic-design/iconography term denotes X-shaped patterns, such as a central type with four type-morphism radii):



Pointcut expression languages extend to designations of type-transformations across different radii in this kind of "semiotic" intertype-space.

Consider, for example, the use of GUI components to interactively visualize values of a given type. For the sake of discussion, assume a type correlated with its distinct GUI class expressly implemented to display values of that underlying type—in this case the relation between the central data-type and the GUI visual-object type is one of directly pairing off one type with another, because an instance of the underlying type is necessary to populate the GUI type. We can then "map" elements of one type to another—in other words, fields in the underlying type can be displayed as text/drop-down selection labels or visual gadgets in the GUI type.

Imagine an EHR form, with labels showing a patient's age, gender (maybe from a fixed selection, such as Male, Female, Trans, Nonbinary, and so forth), first and last name, etc. Data-fields such as "Age" may well map directly between the two types, but even with direct type-to-type correlations, such inter-type mapping can be more complex in general. Suppose, for example, that a EHR form class incorporates a list of the patient's current prescribed medications. Because this list will be of varying sizes for different patients, the GUI representation would have to be not a single label (or some other single-valued visual), but rather an expandable sub-control (such as a list or table display), where the software can add rows to list each of the medications declared by the patient's record. In the underlying data type this information is a *collection*, rather than a one-off value (recall our definition of mereotropic fields in Chapter 5).

Consequently, in the underlying-to-GUI "morphism" a *collection* field must be associated with multiple *procedures* on the GUI side, insofar as visual objects representing each element *within* the collection have to be accounted for—the graphical display for collections needs to expand and be generally modified (for instance, column-widths adjusted, in the case of table displays) to accommodate the variant-sized nature of the originating data source. In short, the underlying-to-GUI association is fully spec-

ified only by enumerating *multiple* procedures and multiple code points—those where the GUI object is modified to accommodate dynamic variations, such as variant-sized collections, in the underlying type such that the identification of these code locations is again an example of pointcut expressions.

In sum, pointcut expressions are intrinsic to rigorous documentation of at least two of the four points (that we have discussed so far, i.e., visual objects and procedural models) in the "semiotic saltire" diagramming how a single data type morphs into different structures to accommodate different coding requirements. Expressing a data type in GUI form is of course relevant for end-user applications, but this is also relevant for data integration, because some data-integration scenarios may require users to expressly examine data values to confirm an integration-computation; a common scenario in cases where, for instance, provisional machine-learning outcomes are subject to confirming human reviews.

Moreover, pointcut expressions can also be applied to *serialization* at least in the context of (what we can call) "grounded" serialization, where serializing markup is annotated with metadata, indicating how the serialization format connects with an underlying type system. A simple example of "grounding" in this sense is the annotation that a markup region serializes a single instance of a specific data type, but more complex grounding declarations can relate a serialization artifact (such as the encoding of a specific data field) to one or more implementation code-procedures (such as the accessors that collectively define the interface for a encapsulated field which can be seen as procedural manifestations of that field, as it is described in a data model, with a single data-model field mapping to multiple accessor-procedures). In this sense serialization-grounding metadata and pointcut expressions share similar procedure-denotational building blocks, aside from the possibility (e.g., via virtual properties implemented through procedure "views") that grounding would reference pointcuts directly.

This discussion therefore suggests that pointcut expressions cut across data-to-code-modeling for three of the four points of the "semiotic saltire." There are scenarios where pointcut expressions may apply directly to *database* logic (the fourth angle of the saltire), but we will argue that the deeper connection between pointcut expressions and database concerns lies in the underlying hypergraph model, as will be explicated in the next full section. Prior to that discussion, however, it is relevant to point out certain additional areas of overlap between pointcuts and procedural interfaces, aside from those already alluded to above, particularly in contexts such as runtime reflection and remote procedure calls.

6.3.3 Annotation-based reflection and procedural binary equivalence

We said earlier that pointcut expressions "locate" points in source code. However, the mechanisms through which such expressions are evaluated may depend on *annotations* applied to the code, so the scope of a pointcut expression language (or a hybrid code annotation and query language which includes pointcuts as one construct) encompasses annotations, which provide the infrastructure for pointcut expression targets, as well as pointcut expressions themselves. Although code-annotations are potentially an intrinsic language feature—Java, C# C++ and other mainstream enterprise-level languages all have some built-in annotation (aka attribute or "decoration") mechanisms (C++ with the caveat that annotations are compiler-specific and not uniformly standardized, although that is gradually changing [26])—this discussion will focus on the (potentially more flexible) use of *external* annotations, where a given code base is accompanied by a distinct code body (potentially in a different language, one unrelated to any

standard or compiler/runtime capability associated with the original language) that describes properties, requirements, metadata, or runtime-usable documentation about the original code. A good case study in the use for such annotations is support for dynamic method calls: providing an application (or in general a code base) the capability of invoking a procedure by expressing a textual description (most simply just the procedure-name) of the desired procedure, along with a textual representation of the desired arguments. Exposing procedures to textual runtime-reflection in this sense has applications for unit-testing, for embedded scripting languages, for fine-tuning the behavior of a running application, and for documenting data-model requirements (because we can dynamically examine requirements declared for different code elements, such as data types, data fields, and procedures, allowing these requirements to be interactively documented as a feature of the associated software application).

To develop the theory of dynamic runtime procedure-calls, consider first that, without runtime reflection, procedures can only be *statically* invoked; in other words, a procedure runs only if there is a point in the source code wherein it is instructed to run. Of course, applications do not in general know exactly which procedures need to be performed to respond to the user's request, since one cannot know what exactly the user will do (the only exception to this principle might be behind-the-scenes programs, such as those which run when an operating system first starts up, but these programs do not typically interact with users at all, and therefore they are not technically *applications*). Most real-world programs, in short, do not follow a fixed sequence; instead, after some preliminary startup, they enter a mostly inactive state (e.g., an *event loop*) and wait for user input. They then respond on the basis of what the user does (clicks the mouse at a given screen location, types on a keypad, etc.).[15] The user's actions are typically called *signals*, and the steps taken by the application to usefully respond to those actions are *handlers*. Handlers generally call different procedures based on the nature of the user's actions, but all logic involved in translating a given action to one or more procedure-calls has to be hard-coded in the application code.

Dynamically invoking procedures via runtime reflection varies in this scenario because the "signal" which the application responds to—assuming the application is in a passive state waiting for input—is not a mouse or keypad event (or other user device), but rather an explicit description of the desired procedure (so the procedure does not need to be inferred by parsing a less-specific input event). For example, when a user clicks on a "save" button, they may very well be unaware that satisfying their intended request requires calling a function called (something like) "**saveFile()**"; but a dynamic runtime-reflection invocation would explicitly designate the relevant procedure (e.g., **saveFile()**) by name. To be sure, in most cases, users would not themselves type in the procedure they want to invoke (although some software, such as **emacs**, works expressly by users typing instructions). On the other hand, one design principle for adaptive user-facing software is to map user-actions and related GUI elements (such as buttons and menu items) to text de-

[15] Neither theories emphasizing concurrency, such as Petri nets or π-calculus, nor conventional lambda calculi seem especially well-suited for modeling program flow in GUI applications. Responses to user actions—which would typically span multiple procedures—may run in parallel (because the user may present new action before a prior response has completes), but the specific modeling challenges in this context are not centered on concurrency *per se*, but rather on indeterminacy vis-à-vis *which* action the user will perform, and the fact that one single overall application state can be affected by any number of possible actions. Perhaps some not-yet-formalized hybrid of (say) λ and π calculi could appropriately model the semantics of GUI applications in this sense. A good foundation might be so-called "object" Petri nets (*see* [23] or [24], for example).

scriptions in lieu of hard-coded procedure calls; a "**save**" button may be mapped to a text string (or even a text file somewhere) containing the instruction "**saveFile()**," which ends up calling that procedure. Using such a text-string as an intermediate evaluation-step would permit the application to be tweaked by modifying the text to call a different procedure, or multiple procedures, as needed. Typically, the user would not modify the text directly, but it could be changed during some sort of application upgrade, or to sync the application with an external cloud service, or some other post-install adaptability feature.

Even when runtime-reflection procedure-calls are not implemented for adaptive purposes along these lines, such capabilities may still be relevant for testing, documentation, and requirements engineering. For example, if a data model stipulates certain data-management requirements (such as unit-scales as discussed above), one can verify that the code base handles those requirements properly by executing a test suite that dynamically calls procedures that are affected by these specifications, verifying that they manage values (and correctly handle malformed data) properly. Aside from double-checking that the code base is properly implemented, the code supporting such testing capabilities provides documentation of the underlying model (according to the principle we earlier referred to as "documentation-through-implementation"). In general, runtime-reflection promotes documentation-through-implementation by allowing documentative descriptions or scripts to manifestly demonstrate features of an underlying data model by invoking procedures where these features are relevant. In short, one strategy for rigorously enforcing data models through code models is to provide external annotations sufficient to allow data-model-sensitive code to be tested via external invocations.

To be sure, external invocation requires more than just describing a desired procedure: the runtime-reflection code needs to parse some encoding of arguments *to* the procedure, and moreover this runtime-reflection code needs to set up the procedure call so as to *mimic* the application binary interface (ABI), which collectively represents the compiler, operating system, and programming-language specifications for how procedures may be called at the machine-language level (this is less applicable to dynamic languages, such as Lisp, or indeed languages such as Java or C# with built-in reflection capabilities, but it applies directly to lower-level languages, such as C and C++).

We'll concentrate on C++ in particular: there are several different strategies for supporting C++ reflection, each with distinct trade-offs. One powerful solution is to use an expansive runtime-metadata framework (particularly LLVM) which allows almost any procedure to be exposed to a reflection system (and therefore invoked dynamically from an external call-description). The problem with libraries, such as LLVM, is that they tend to make code bases much more difficult to compile and install than the equivalent code base on its own, which limits the feasibility of distributing the entire code base in source-code fashion.

A different solution is typified by Qt (the most widely used cross-platform C++ application-development framework), which relies on preprocessing certain source-code files; this approach is limited by the necessity of that preprocessing step and also by the restriction that only procedures (and likewise data-fields) that fit certain technical criteria may be exposed to the Qt reflection system.

The AngelScript embedded scripting language is a good example of a third alternative, where the language runtime translates certain runtime logic directly to assembly language, adapted for a number of different operating systems and compilers; the limitation of such a solution is (first) that such code becomes, for all intents and purposes, completely opaque for any programmers who are not intimately famil-

iar with multiple target platform's assembly instruction set, and (second) that it demands fairly complex programmer-supplied code to describe exposed data types (as well as procedures) to the AngelScript runtime.

A fourth scenario is represented by Lisp dialects such as ECL (embeddable Common Lisp) [3] and Clasp [48], where C++ functions can be exposed to Lisp code via a "foreign-function interface" (or, at least in Clasp, can potentially be compiled directly into the Clasp system as hidden procedures in object-file scope). These approaches require that C++ bridge code be able to work directly with Lisp constructions, such as pointer-fixnum unions and cons cells, which leads to very non-standard-looking C++. Such problems are characteristic of the general approach taken by most C or C++-interoperating scripting languages, which is either a fifth alternative (when enumerating C++ reflection strategies) or a variation on code pre-processing; namely the implementation (either manually coded or automatically derived via a preprocessing tool) of "wrapper" procedures, which marshal data back and forth between C++ (or, likewise but less complexly, C) and the relevant scripting language (Python, Ruby, JavaScript, etc.).

In this book we are proposing (and illustrating in prototype fashion) a framework for general code annotation and pointcut-expression declarations/evaluations, which includes a system for resolving procedure-descriptions to ABI-compatible invocation structures at runtime. While any of the strategies just enumerated could potentially be applied to such a framework, we propose a system based on the notion of *procedural binary equivalence*, which eliminates almost all of the problems associated with the above-mentioned reflection paradigms, while significantly reducing the programming effort (and boilerplate code) needed to provision applications with runtime-reflection capabilities. The central observation for this technique is to note that from an ABI perspective the machine-level code is not concerned with the actual types accepted by a signature, but simply with the binary profiles (the byte-lengths and, potentially, constructor/destructor/cast functions called on temporary values) of values conformant to those types. In other words, a pointer to a function of one type can safely be cast to a related function-pointer-type, so long as the types constituting the latter type's binary and temporary-value aspects are the same as the actual types of the targeted procedure. Via such equivalences, the expansive range of types that enter into procedure signatures can be scoped down to a much smaller collection of "canonical" types, or (the terminology we propose) "pretypes," so that we construct equivalence classes of many different functions that are distinct on the type/signature level, but binary-equivalent on the "pretype" level. Exposing a procedure to runtime-reflection thereby entails simply classifying the procedure within one of the binary equivalence classes.

This technical framework comes with some caveats. First, binary equivalence is (to some extent) compiler-specific, so adopting these techniques involves either knowing *a priori* that a code base is specifically targeting a given compiler (GCC, say) or else using runtime and/or compile-time checks. In addition to compiler variation, binary equivalence may also depend on factors endemic to the code base, such as whether smart pointers are utilized, or whether certain specific data types are prominent (such as **QString** or **QVariant** for application composed via the Qt framework); the actual binary equivalence-space may accordingly vary from one code base to another.

A second caveat is that combinatorial expansion of the number of distinct binary equivalence classes needed to support relatively fully flexible procedure-exposure can potentially make compile-times too long. It is, in short, unrealistic to expect *every* procedure in a code base to be directly invoked using techniques described here; for procedures with unusually

complex signatures, it may still be necessary to provide wrapper code (e.g., instead of a function that takes variant-length argument packs provide a version taking a reference-to-vector, expose this latter function to reflection and call the former function from the latter). Nevertheless, binary-equivalence techniques should make it possible to greatly reduce the number of occasions in which wrapper code needs to be written or generated. For procedures whose signatures fit within a given binary-equivalence scheme, exposing the procedure to a runtime-reflection engine would then be as simple as annotating the procedure with a single numeric code matching the procedure to an equivalence-class. The practical details of such annotations are outside the scope of this chapter, but are documented in some detail in the book's supplemental materials.

With such a reflection system in place, it is possible without significant modification to provide runtime-reflection to a code base, particularly one implementing a "documentation-through-implementation" project, wherein individual procedures (and their properties declared as code meta-data) instantiate and exhibit data-model specifications. Such an approach raises the possibility of examining code in the vicinity of specific procedure-calls to verify and/or document the connections between code and data models—i.e., test that the code base accurately instantiates an underlying data model and/or document the data model by expositing its implementation in the code. An example of such documentation might follow from the above discussion of unit-scale verification: the code's logic for checking scale-measure properties on input/output values and lexically scoped variables could be tested by externally simulating calls to procedures that input scale-delimited values as well as calls to the outer procedures that call such procedures in their function body. In general, an outer procedure may go through several steps to ensure that parameters passed to a called procedure are in accord with data-model requirements. A useful design pattern is to implement this preparatory logic in anticipation of the code being verified/tested by external runtime-reflection calls; here both outer and called procedures would be exposed to the reflection engine and, moreover, a logging or documentation mechanism could be implemented for the code *surrounding* the inner procedure-call to clarify steps taken to ensure data integrity.

6.3.4 Meta-procedural, procedural, and sub-procedural syntagmatic scales

The above discussion points to a larger topic in the overall theory of modeling computational processes: a computational step that functions logically as a single procedure-call may encompass several additional steps before or after the actual procedure-call involved, as data entering (and returning from) that procedure is assembled and/or validated. This expanding-outward phenomenon is particularly evident in the case of *remote* procedure calls, which are more complex than simply temporarily delegating execution to a different subroutine than the one currently running. A procedure that invokes a different procedure on some remote service (a cloud service, say) may be unable to simply pass raw values (there is no ABI binding remote procedures), but instead may need to encode inputs in (say) a textual markup format, and similarly deserialize data received as response. More often than not the calling procedure does not wait for the remove call to return, but rather provides an anonymous procedural value to act as a return-handler. So in a typical scenario, remote procedure calls involve data serialization/deserialization and intermediate procedural values (moreover, on the remote endpoint, clients do not typically request actual procedures directly but invoke *services*, or what may be called *meta-procedures*, which perform data-marshaling and delegate to *de facto* procedures based on the provided request). In short, remote procedure-calls involve prepara-

tory code on both endpoints, which serve as a logical framing of actual procedure calls with additional data-marshaling/handling.

As noted above in the context of Syntagmatic Graphs, a single procedure-call is built up in stages, even though from the perspective of high-level programming-language syntax a call can be expressed as one single expression-unit (the intermediate stages being largely invisible in the surface-level code, but implicit in the compilation of this code to machine and/or virtual-machine representations). On the other hand, as the examples of scale-measure checking and remote-procedure calls demonstrate, sometimes the multi-stage framing of a call propagates to computational steps that *are* explicitly visible in high-level code, and sometimes (as in remote-procedures) the logic actually involves *meta*-procedures, which augment the structures of procedure calls themselves with extra data-marshaling and handling logic.

Formally representating the temporal sequencing involved in performing and managing a single procedure-call can therefore concern several different levels of code "granularity," that "below" the original source-language (as in compiler-related intermediate representations) and "above" (as in networking with remote-procedure providers) as well as on the level of the source-language. This is why we propose a sufficiently expressive code-representation system (e.g., syntagmatic graphs) to embody coding structures at each of these levels. By extension, code-validation via runtime-reflection can test/document coding assumptions by testing implementations at each of these three levels. This is a further rationale for using code-modeling as a data-modeling device, and also points to another feature that a code-annotation language should support: the capability of expressing code structures at the three levels (from what we might call "sub-procedural" to "meta-procedural") described here. These represent different coding environments where the "procedural" aspect of the "semiotic saltire" would be in effect.

In effect, the general model of computation that we are operating with here—where the lambda-calculus notion of "reduction" is replaced by a process-calculus-like "binding," implying a connection between descriptions of computation and epistemic semantics based on "information content"—this model can be manifest at several different levels or scales of computations, insofar as these are identified in computer code. We'll call these subprocedural, procedural, and meta-procedural *syntagmatic scales* so as to have convenient terms for such levels. Again, the main goal of introducing and arguing for special terminology is to establish structural features that will be useful for query languages; insofar as a pointcut expression language can be incorporated as one part of a hypergraph description/siting language, terms for different syntagmatic scales can become terms used and recognized in the query language.

6.3.5 Case study: annotation and image markup

This section will conclude by outlining how themes discussed in this chapter can be demonstrated via one specific data format and code library, namely the Annotation and Image Markup project (AIM) [7], [40], [46], [49], [29]. Originally part of the cancer Biomedical Informatics Grid (caBIG), AIM standardized the encoding and sharing of image-annotations used by radiologists and other pathologist to diagnose conditions (tumors, for example) from medical images. An image-annotation in this context is typically a geometric description or designation of a "region-of-interest" (RoI), which the examiner believes is diagnostically significant. Regions-of-interest can be demarcated in simplified geometric forms (via circles, rectangles, and so forth) or isolated by creating overlays, which themselves have an image-based format (the simplest example being a black-and-white picture, where

white pixels represent the region/foreground and black the background), so as to represent an RoI more granularly.

At present we will initiate merely a brief overview of AIM and image-annotation concerns, deferring to subsequent chapters to continue our analysis in greater depth. This book's supplemental materials will republish code within the AIM C++ library (AIMLib) with minor changes for modern compilers (the original code was developed around 2013), including parsers for consuming AIM XML as well as demo files obtained from The Cancer Imaging Archive (TCIA). More substantially, we provide a client library to simplify the process of loading annotations as runtime objects from AIM serializations. This AIM client library (published as an **aim-client** Qt project), which wraps some of the functionality exposed by AIMLib, hopefully demonstrates how specifications within data models can be translated to requirements or functionality implemented within code models. The **aim-client** code provides several forms of runtime-reflection which supplements the underlying AIMLib code base, such as dynamic procedure-invocation and also type-reflection capabilities. For example, **aim-client** packages standard C++ **typeinfo** functionality into an AIM-specific type-reflection class that manages details such as type-name demangling. A typical use-case for type-reflection, demonstrated in the book's accompanying code, involves casting an abstract **MarkupEntity** pointer (a base class that is specialized for different annotation-shapes) to its specific annotation type. Such reflection capabilities simplify the process of extracting the core annotations from the full suite of details encoded via an AIM annotation collection (specifically, an instance of the **AnnotationCollection** class). The demo archive also provides extensions to several existing 2D or 3D software applications that can furnish origin-points for images to be annotated, along with a GUI-based annotation program; coding constructs within these components serve to illustrate certain themes discussed in this chapter—such as, identifying interrelated procedure-groups (e.g., for reflection)—with these points discussed via source comments as appropriate.

Bioimage annotations intrinsically represent three different sorts of information: geometric data constituting annotations themselves; presentation data governing how annotations are made visible to human users; and bioinformatic content, which provides an interpretive context for the image data (e.g., a description of the specific tissues, organs, or organ systems visible in a bioimage). One step in semantically organizing the totality of annotation information is to clarify which data points serve which roles. For example, in the **GeometricShapeEntity** class, data fields such as **LineColor** (and line thickness, opacity, style, etc.) are presentation details, whereas the annotation's vertex coordinates (a sibling class to **GeometricShapeEntity**) are underlying spatial data (logically separate from any visualization). The **aim-client** library uses several techniques to render these semantic details more explicit, relative to the underlying AIMLib code; for example, we provide stronger typing for geometric and presentation details (colors and line-graphics specifications are objects rather than C++ strings, for instance).

In the following chapters we will analyze in greater detail both AIM specifically and the data models associated with image-annotations in general. In particular, this topic provides concrete examples of how data-modeling issues often become manifest in terms of code-modeling decisions (such as procedural pre- and post-conditions) and logical relations amongst implemented procedures, which has been a central theme of the current chapter.

6.4 Hypergraph representations for data-persistence bridge code

This chapter's discussion has addressed the features and affordances of hypergraph data

modeling, particularly via syntagmatic graphs and pointcut expressions, in three of the four "radii" of the "semiotic saltire": inter-type connections for GUI components paired with underlying data types; "grounding" for underlying types' serialization; and syntagmatic graph models of procedure-call logic at meta-procedural, expression-level, and subprocedural level (fully elaborating a theory of these three levels would require additional discussion outside the scope of this chapter, so we leave the terminology not-too-rigorously defined at this point, but hopefully examples such as remote procedure calls, verifying scale-unit consistency, and bindings/temporary objects accumulating data for procedure-calls, respectively, illustrate the kinds of computational phenomena applicable to these three levels). This final section will address hypergraphs in the context of the fourth saltire aspect (data persistence).

As earlier in this chapter, we assume that data and code models are tightly interconnected—specifically, that data models are instantiated by code libraries, wherein data types that are intrinsically or schematically described in a data model become explicitly implemented in data types coded via programming languages, particularly general-purpose languages such as Java or C++. As such, data *persistence* can be defined as the process of storing typed values that are instances of types implemented in a code base so that those values may be used in the future (and/or shared with other applications employing the same code base). As intimated above, there need not be a direct isomorphism between databases and implemented types—for instance, it is not necessarily the case that every table in a relational database can be paired with a type whose fields correspond to the table's columns—but treating databases as persistence engines for a project's data types is a useful perspective from which to examine data modeling, sharing, and representation patterns.

The correspondence between databases and implemented data-types is especially strong in the case of *object* databases, which are designed so that almost any values used by an application can be directly stored in a database, with little extra code needed to marshal data between different representations. Although object databases have been explored for at least a quarter-century, they have not emerged as mainstream architectures comparable to either SQL (relational) databases or predominant NoSQL models. Perhaps the mismatch between live program memory—how data is represented while an application is running—and the structure needed to efficiently persist and then retrieve data, evinces a gap too wide for object databases to serve as viable replacements for SQL or for the most popular NoSQL architectures [4, pages 6*ff*].

Instead, the optimal database architecture probably lies in between the rigidity of the SQL model, where the structure of information within the database is significantly different from the natural structure of programming-language data types, and object databases, where database structure tries to directly mimic in-memory data layouts. Successful database systems find a reasonable balance between employing structures that are conducive to efficient data storage and query-evaluation and, at the same time, minimizing the degree of difficulty apparent in the translation of application-level data types to data stored within the database itself.

Concerns about data *layout* and *query evaluation* have the consequence that application data must be structurally transformed to be stored in a persistent database. It is worth pointing out the specific complications introduced by layout and query-evaluation concerns to motivate database engineering solutions to the problem of bridging application data types to in-database structures. A database, first and foremost, holds data values for an indefinite amount of time, but such data persistence is only one part of its full suite of roles and requirements.

In and of itself, constructing a persistable representation of most data types is not especially

difficult: most programming environments have some sort of "binary stream" encoding, where numbers are directly represented in their binary form, character strings can be encoded by assigning a number to each distinct symbol in the strings' character set, and composite structure can be encoded by encoding each of their parts (e.g., date-times can be encoded as a single value, such as milliseconds-since-epoch, or as a tuple of second-minute-hour-day-month and so forth). In Qt, for instance, the **QDataStream** class can serialize most data structures in binary form by passing individual fields into the "stream" (e.g., **qds « person.name() « person.age()** and so on for a **qds** datastream and hypothetical **person** class). Using classes such as **QDataStream** (or their equivalents in other languages/environments), it is not difficult to create binary packages representing most or all of the data in a given typed value, so that this binary encoding may be stored within a database and eventually reconstructed as a duplicate of the original value, yielding data persistence without much programming effort. At this level, then, data persistence does not require a complex database architecture.

There are two main problems with this simplistic approach to persistence: first, the resulting data storage might not be *efficient*, with the consequence that individual values take up more memory-space than needed, which can be a problem if a database extends to the scale of millions (or billions) or distinct values; and, second, the problem of *finding* individual values from a large data-collection. It is one thing to have some value in application memory while running the application on one occasion; it is a different matter to re-find that value (maybe years) later amongst millions of other values.

When values are binary-encoded via classes such as **QDataStream**, the data within the stream is, in general, "opaque" to the persistent database engine: the engine cannot "read" values within the stream, and from that perspective the stream is only a sequence of bytes with no specific meaning. In a working database at least some content associated with a given aggregate data-value must be exposed to the larger engine; if a value represents a *person*, say, the engine would probably need to identify the person's *name* so that one could find a person's record if one knows their name in advance. In other words, although *some* of the data in a given application-level typed value may be exported to a database simply as an opaque binary stream, at least some part of this data must be expressed in a queryable fashion, one which is tractable to the surrounding database engine.

Accordingly, with an instance of an application-level data type—or in general a type implemented within a code base associated with a corresponding data model—one desideratum for database persistence (which can be formally addressed in the data model) is how to divide the data within each specific type into "opaque" data, which does not need to be queried (outside the context of a fully instantiated instance of the data type) and *queryable* data that might be used to locate that specific value against many others. For instance, in an EHR database almost certainly a patients' first and last name would need to be queryable, because it is normal to attempt to locate a person's records by providing their full name (if a patient calls their doctor they are more likely to identify themselves by name than by some in-hospital ID number, for example).

Queryable data, moreover, can sometimes take the form of individual-value fields (such as first and last name), but can also take the form of collections. For instance, given an EHR database that tracks medications, it is reasonable to form queries based on the one-to-many relationship between a patient and the medicines they have been prescribed; e.g., to list all patients who are currently taking a particular pharmaceutical. Such a query can only be evaluated if the list of medicines prescribed to each patient is queryable in the system (and not opaque data

that can only be read by reconstructing a whole patient-object at the application level).

In short, information stored in a database can be divided into three groupings: *opaque* (non-queryable) data, non-collections queryable data, and collections (multi-valued) queryable data (which we might call "multi-queryable"); each of these forms of data have different layout and management requirements. An effective database engine will store data in a concise manner (to use as little memory as possible) without making queries inefficient (it must be possible be locate queryable data fields relatively quickly). Different database engines use different techniques to optimize memory-layout; the details of such optimizations are outside the scope of this chapter (and are not the direct concerns of application code in general), but we can point out that application data models do need to identify which data fits into which medium (opaque, queryable, and multi-queryable).

Different database engines, aside from identifying "queryable" data (typically without actually employing this term), also have different strategies for *relating* queryable data to its associated overall data-type instance. For example, consider the process of locating a patient record using the patient's full and last name. In SQL, the two-column conjunction of first and last name might serve as a compound key uniquely identifying patient-records in a patient table. In a property-graph database, the compound first-and-last-name value may serve as a property annotating a patient-node, utilizable to locate that specific node (potentially within a large graph will many other patient nodes). In an RDF graph database or "triplestore," the combination of a fixed last-name and first-name node-value (together with "last-name" and "first-name" as edge-labels) might provide "shape constraints," which filter nodes down to a unique match yielding the desired patient-node. In short, the graph and/or tuple-structures and terminology that govern how queryable data serves as an index to retrieve records from a database differs according to the underlying database architecture.

A given application and/or code base therefore has multiple options with regard to the requirement of modeling how a data-type instance should be registered in a database in queryable fashion. Queryable data may be exposed as a *property* (uniquely identifying an overall type-instance), as a node within a hypernode (in the case of hypergraph databases), as a foreign-key or an index value (in a relational context), and so forth. The specific structures and terminology will depend on the specific database that the application uses for data-persistence. However, abstracting from the details of different architectures, we can consider an *abstract* persistence model which would recognize structures such as properties and hypernodes as generic patterns that can be translated into the architecture-specific forms of different databases as a concrete instantiation of a more general abstract model (see Fig. 6.4).

Presuming this possibility, then, we propose the notion of a general-purpose *persistence model* which fits into our earlier "semiotic saltire" picture, where the persistence for an individual data type would employ properties, hypernodes, queryable and multi-queryable values, and similar formations to construct a representation of how data type-instances should be encoded when represented in persistable form. This persistence model can be thought of as a data-structure recipe targeting a "virtual" database engine that incorporates features of specific database architectures, such as property graphs and hypergraphs. By expressing data-persistence logic in this generic fashion, the data-model thereby includes as one part a persistence-model which provides an abstract picture of how types within the data model should be persisted (taking memory-layout and queryability concerns into account). Specific code libraries could then translate this abstract persistence model into concrete representations suitable for whichever specific database engines

may be used as back-ends. (One benefit of this generic model is that the persistence logic is not tied to a specific back-end; the same abstract model can be compatible with many different back-ends, so that different back-ends may be selected as application evolves and different database features become more important; for example, over time, it might be necessary to select a new database engine that places greater emphasis on scalability, or on automated backup and replication).

Earlier in this chapter we suggested that a *pointcut expression language* can be grounded in hypergraph semantics in the sense that, insofar as source code is modeled via syntagmatic graphs, pointcut expressions become, in effect, one form of hypergraph site-location/traversal query. In the context of data persistence, according to the "abstract" model just presented, we similarly suggest that a *bridge language* for representing data type-instances in pre-persistence forms is similarly grounded in hypergraph semantics in that these bridge representations are an example of hypergraph serializations. In this sense, such a pre-persistence language (referring to representations that constitute an intermediate step derived from in-memory data, but from which in-database structures can subsequently be constructed) can be paired with pointcut expressions and grounded-serializations as representational concerns (relating to different radii on the semiotic saltire) that all translate to hypergraph queries (and therefore can be implemented in terms of a common hypergraph-query parsing and evaluation engine). These three paradigms (pointcut expressions, grounded serialization, and pre-persistence bridge forms) are three aspects of the prototype hypergraph-query language that this book examines as a case study in data-integration tools.

As just laid out, bridge pre-persistence is distinct from pointcuts and runtime reflection in general (despite their common hypergraph background). However, the data-modeling issues that should be addressed when selecting a pre-persistence scheme for individual data types does overlap with code-model concerns that would be exercised via pointcut expressions. For example, annotating data fields as queryable or multi-queryable tends to impute on those fields semantics which are also apparent in construction patterns for the type in question (e.g., a field which uniquely selects a given type-instance in a query environment where many such instances are present—one might call such an environment "crowded"—may well be a field whose value cannot be default-constructed, so that some mechanism should be in place for indicating that fully constructed instances must initialize that field in particular). Likewise, annotating a field as multi-queryable implies that it is a collections formation with a set of state-management procedures (insertions and deletions) whose pre- and post-conditions would be specified by a rigorous code model. All told, then, in practice, design patterns for pre-persistence and pointcut-expression use-cases for their common hypergraph description/query language would tend to intersect.

6.4.1 Multipart relations with roles

Another facet of pre-persistence involves optimizing the database layout to represent relationships between relatively complex objects; that is, between objects which have enough internal structure where it is unlikely that one would usefully be encoded as a single field or property inside the other. In relational databases, this is the kind of situation addressed by foreign-key references linking two tables. In graph databases, such scenarios lead directly to labeled edges connecting two nodes (or hypernodes). In principle, any node in a graph database can be connected to any other node, so it is straightforward to connect any two data-values with however much degree of internal structure they may have, so long as a single node (or hypernode, in hypergraph contexts) encapsulates or delegates reference to each value respectively.

A familiar problem in graph-database data modeling, however, concerns *multi-part* relations, where fully asserting all details of a given instance of the relation requires naming more than two values (or objects, nodes, etc.). A popular example for examining multi-part relations is how the familiar binary relation of *marriage* can be generalized to a more complex structure if we identify details such as *whom* (officially) married the bride to the groom. Similarly, a *divorce* is a multi-part relation that (minimally) includes a husband and wife, but also requires a marriage contract of some sort that is voided by the divorce-event (and therefore such details as a marriage date, with the divorce then marking a temporal range when the marriage was in effect, followed by a time when it is annulled; plus perhaps details confirming the prior marriage's legal status, because only an "official" marriage can be superseded by a divorce). For the sake of discussion, consider marriage and divorce dates, as well as identifiers for the officiating parties who certified the marriage and the later divorce, as added details marking *divorce* as a multi-part, rather than binary relation.

Different data-modeling strategies address multi-part relations in different ways. In general, the relation in question can either be treated as a compound, whose individual components are distinguished in terms of the roles they play in the larger whole, or as a data structure that functions as a peer to the data-values connected by the relation. In a graph database, say, where nodes represent "values" or "objects" and edges represent relations, multi-part relations can be modeled as *nodes*, in which case the elements participating in the relation would be represented in accord with any other node-structure, as data fields or nested-nodes encoded via whatever structuring representations the database architecture supports (properties, hypernodes, shape-constrained neighborhoods, etc.). For example, we would have a "divorce" node whose data-fields represent the relevant husband, wife, marriage date, divorce date, certifications, and whatever related data; nodes representing the two spouses would then be connected to this divorce-node to model the spouses being divorced. The fact that divorce is also a *relation*—we can say that John *is divorced from* Jane—is then implicit in the existence of the "divorce node" to which both parties are connected: the structure does not explicitly notate this relation as a connection between, say, the *Jane* and *John* nodes.

Alternatively, the notion of multi-part *roles*—which is a prominent feature of graph databases targeted at AI applications, such as **Grakn**—tries to more directly model what we perceive to be the real-world semantics of multi-part relations, where *divorce* (say) is indeed a *relation* between two parties, even as it has additional structure. We can accommodate the extra structure by noting that different parts of the relation fit into the larger whole in different ways: so the divorce has one part which is a husband, one part a wife (of course same-sex divorces are a also possible; we speak in gendered terms simply to refer conveniently to the distinct parties), other parts are dates and certificate IDs, and so forth. The complete relation is semantically the totality of individuals or details playing each of these implied roles.

In effect, multi-part relations—which we will refer to for analytic purposes as "multi-relations"—can either be modeled by *role-sorting*, which ascribes distinct roles to different participants, or *reification*, which converts a relation that would ordinarily be represented as a *connection* between two objects (an edge) and converts it to an object in its own right (taking here "object" to mean any compound data structure with its own fields to be graph-represented). The most semantically accurate representation is arguably to combine these two options. Consider the function of multi-part relations from the perspective of graph queries and traversals: suppose I know *John*, and wish to learn the name of his ex-wife (or analogously a traverser is on the John-node and wants to visit *her*

node). The divorce-relation should take me between those two nodes; this expectation remains in effect, even if the relation is multi-part, because a multi-relation should still serve as a fact authenticating a step between two related nodes on the basis of that relation being known as a fact.

In other words, a multi-relation should supply traversal steps no less than would a binary relation; the difference is that multi-relations enable *several* traversal paths, and we can use *roles* to select one direction or another. Getting to *Jane* from *John* traverses the *divorce* relation, but specifically the *spouse* role embedded in that relation. Invoking other roles (say, marriage-date) would cut across the multi-relation in an alternative direction. From the perspective of queries and traversals, accordingly, the multi-relation behaves functionally like a set of binary relations: divorce encompasses *ex-spouse*, *marriage date*, *divorce date*, and so forth. However, these implied binary relations are not independent graph elements; instead, the multi-relation enforces interdependencies among them. If (say) a correction were to update the *divorce date* value, this would necessarily alter the implicit relations identifying *John*'s and *Jane*'s divorce-dates, respectively.

Semantically, then, multi-relations could most accurately be described as interconnected sets of edges (or structures that logically play a role akin to edges); i.e., from one multi-part relation there is a collection of different traversal-steps that can be taken via the data encoded in the relation as an authenticator (on the premise that knowledge-graph facts "legitimize" steps between nodes when one is seeking to obtain information constrained by known facts, e.g., to get information about an ex-spouse starting from the other former spouse). However, these implicit step-possibilities are not "autonomous" as they would be with a disparate collection of edges, instead of one multi-relation that logically implies those edges; changes to node-values related to one part of the relation may potentially propagate to other parts and participants in the relation. In this sense the multi-relation operates akin to an aggregate of "reactive values" in functional reactive programming, where state-changes triggering updates in one value may (as an implementational requirement on the reactive engine) cascade to updates on the other values (*see* e.g. [44] or [27]).[16] Consider calling a collection of edges, which are interdependent along these lines (in the sense that node-value changes in the neighborhood of one edge may propagate to others), an "edge tangle" (see Fig. 6.5). A multi-relation is then in effect a data structure which merges the semantic and operational roles of *edge tangles* and of *relation-reifying* objects (in effect, the multi-relation object simultaneously reifies each edge in the edge-tangle).

This chapter has concentrated on procedure-calls more than on graph structures, but the two notions are interrelated, an idea which is especially apparent in the case of multi-relations. In

[16] A *reactive expression* is a kind of union between a single value and a stored-expression which yields a value upon deferred evaluation, with the specific property that the value updates whenever the value of the stored expression would change due to modifications in its components (akin to a deferred expression that can be re-evaluated arbitrary many times, rather than only once, but canonically a side-effect-free expression that should only be re-evaluated when its components have changed). For example, a deferred expression in a GUI context might be the aspect-ratio of a GUI window, which gets updated when the window is resized. Reactive expressions thereby "tangle" their components (in the window case, those would be the window's width and height, and/or corner coordinates). Reactive expressions are bound to object-state (for some updatable object, e.g., a GUI components) with the idea that a reactive engine uses those expressions to maintain certain constraints (such as GUI layout constraints), even in response to external change (which the engine reacts to); declaring constraints in terms of reactive expressions obviates the need to implement procedural logic which enforces those constraints via callback handlers. Multi-relations could potentially be seen as akin to reactive expressions that provide satisfaction for constraints implicit in the coexistence of component relations in the single multi-relation; for instance, if John has divorced from Jane, then John's divorce-date is constrained to be the same as Jane's.

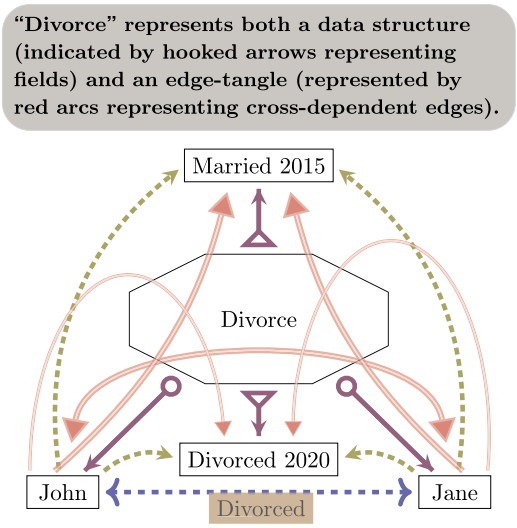


FIGURE 6.5 Multi-relations and edge-tangles.

graph modeling, a relation serves to guide or legitimate a traversal step; in some contexts, a relation may provide operational possibilities effectively akin to a function (e.g., a procedure): using a graph-edge to obtain the node of an ex-spouse from that of the other ex-spouse is analogous to calling a procedure that outputs the one ex-spouse when passed the other. Stepping from one node to another by following a constrained traversal strategy is, in short, not unlike calling a function that inputs the origin node and outputs the destination node.

Pursuing this analogy: a multi-relation acts like a function that can branch in different ways (e.g., by indexing branch options according to disparate roles); in this sense multi-relations are effectively dual to single-return procedures. A procedure (in general) takes many inputs and returns one output, whereas a multi-relation (in the context of a single traversal or query) takes one starting node and presents multiple possible traversal directions.

The fact that *roles* can be used to differentiate multiple destination-paths for a multi-relation suggests the dual idea that roles can similarly index different *input* sources for a procedure. Such an idea has not been pursued in full generality by mainstream programming languages; named (rather than positional) parameters and distinctions such as "**this**" objects from ordinary arguments (which we have modeled in terms of "channels") are rudimentary examples of parameter-role systems, but a more complete such system would support user-defined roles responding to the semantics of the target domain (rather than just the semantics of the programming language). However, parameter-roles can certainly be declared through a code-annotation system, and therefore externally introduced and runtime-reflected using pointcut semantics, as discussed earlier in this chapter. In this regard, parameter-roles can both add further expressiveness to external code-reflection and can leverage the idea of procedures and multi-relations being in some sense dual (and thereby potentially subject to similarly structured metadata). We will discuss this aspect of parameter-roles further in Chapter 9, particularly in the context of Conceptual Spaces. We will argue that attempts to model procedural semantics via "blended" Conceptual Spaces, which is essential to Coecke *et al.*'s strategy for employing Conceptual Spaces as a grounding semantics for simulations of natural language, call strongly for the structural details afforded by parameter-roles.

6.4.2 Syntagmatic graphs and conceptual spaces

This chapter has, to varying degrees of detail, examined pointcut expressions, pre-persistence bridge representations, and "grounded" serialization as three different aspects of data-and-code-modeling, which can be subsumed within a common hypergraph-based description/query language. Each of these representational use-cases come into play in different coding concerns that need to be addressed when developing a robust code model that can cover the

various dimensions of a full-fledged application (or code base that may be used by one or more applications), such as database persistence, serialization, and GUI integration. We suggested that these variegated coding concerns can be schematized by the visual "saltire" device, connecting a central data type to associated types and/or representations which track how the type is restructured to accommodate different computing environments (e.g., databases, GUIs, and network-based data sharing, where serialization/deserialization protocols come into effect). This picture illustrates how hypergraph-based codes and data representations can be exported to these various aspects of the overall application-development process.

In particular, pointcut expressions, pre-persistence, and serialization, according to the schemas we suggest here, all have a hypergraph basis, where query syntax and evaluation reduces to various forms of hypergraph site-locating and traversal. In other words, there is an underlying hypergraph semantics which provides the structures leveraged by the relevant hypergraph-query functionality. This semantics has not been formally set forth here, but a prototype of a semantic model adhering to the principles we have outlined is provided in the accompanying code, which can hopefully serve via demonstration-by-implementation as a useful proxy for a mathematical exposition of the underlying semantics.

Since the initial form of this semantics was also presented in the context of Coecke *et al.*'s work on hypergraph categories, it is reasonable then to consider how the hypergraph semantics underlying our pointcut/pre-persistence/serialization schema can be related to the conceptual-space-based semantics marshaled by Coecke *et al.*'s formulation. For example, we summarily discuss paths in syntagmatic graphs as tracing increases in information content, or "accretion of detail," which lead toward satisfaction of preconditions for procedures (or verbs, in the natural-language context) as prerequisite for procedure calls. The syntagmatic constructions underlying such a path-model is analogous to the syntactic constructions analyzed by Coecke *et al.*: in effect, syntagmatic paths are analogous to morphism-chains in hypergraph categories.

In Coecke *et al.*, however, an essential point of their analysis is that there is a tight coupling between paths in the syntactic and semantic sense: morphism-chains implicitly "carry," in the sense of compelling and serving as a map-image for, structurally resonant paths in (some form of) conceptual space. The analogous semantic notion on our account would be "expansions" of information content, but this is a less structurally detailed construction of *semantic* paths than Coecke *et al.*'s model. As such, we can consider whether (and how) to adopt conceptual spaces as a source for a more rigorous "path semantics" to mirror our syntagmatic graph syntax, by comparison to how Coecke *et al.* use conceptual spaces to provide an image-domain (in a sense to mirror) morphism-chains in hypergraph categories. One difference between the two models is the nature of the hypergraph categories involved; Coecke *et al.* work with straightforward monoidal structures whereas hypergraph semantics in our context involve query-processing states for hypergraph query languages, so the hypergraph basis on our end has more structure (although for reasons just outlined the path-semantics side, at least so far, has *less* structure). We will consider these comparisons in greater detail in Chapter 9.

References

[1] Benjamin Adams, Martin Raubal, A metric conceptual space algebra, in: International Conference on Spatial Information Theory, Proceedings, 2009, pp. 51–68, https://pdfs.semanticscholar.org/521a/cbab9658df27acd9f40bba2b9445f75d681c.pdf.

[2] Benjamin Adams, Martin Raubal, Conceptual space markup language (CSML): towards the cognitive semantic web, in: IEEE International Conference on

Semantic Computing, Proceedings, 2009, pp. 51–68, https://ieeexplore.ieee.org/document/5298627.
[3] Giuseppe Attardi, The embeddable common Lisp, ACM SIGPLAN Lisp Pointers 8 (1) (1995) 30–41, https://common-lisp.net/project/ecl/static/files/papers/ecl-1995-attardi.pdf.
[4] Sikha Bagui, Achievements and weaknesses of object-oriented databases, Journal of Object Technology (2003), http://www.jot.fm/issues/issue_2003_07/column2.pdf.
[5] Lucas Bechberger, Kai-Uwe Kühnberger, Measuring relations between concepts in conceptual spaces, in: SGAI International Conference on Artificial Intelligence, 2017, https://arxiv.org/pdf/1707.02292.pdf.
[6] Walter Cazzola, et al., Semantic join point models: motivations, notions and requirements, in: Software Engineering Properties of Languages and Aspect Technologies, Proceedings, 2006, https://hal.inria.fr/inria-00542782/document.
[7] David Channin, et al., The caBIG annotation and image markup project, Journal of Digital Imaging 23 (2010) 217–225, https://pubmed.ncbi.nlm.nih.gov/19294468.
[8] Bob Coecke, The mathematics of text structure, https://arxiv.org/pdf/1904.03478.pdf.
[9] Bob Coecke, et al., Interacting conceptual spaces I: grammatical composition of concepts, Extended version of Proceedings of the 2016 Workshop on Semantic Spaces at the Intersection of NLP, Physics and Cognitive Science, pp. 11–19, https://arxiv.org/pdf/1703.08314.pdf.
[10] Ralph Debusmann, et al., A relational syntax-semantics interface based on dependency grammar, in: 20th International Conference on Computational Linguistics, Proceedings, 2004, https://www.aclweb.org/anthology/C04-1026.pdf.
[11] Keith Devlin, Situation theory and situation semantics, in: Handbook of the History of Logic, vol. 7, 2006, pp. 601–664, https://web.stanford.edu/~kdevlin/Papers/HHL_SituationTheory.pdf.
[12] Juneki Hong, Jason Eisner, Deriving multi-headed planar dependency parses from link grammar parses, https://www.cs.jhu.edu/~jason/papers/hong+eisner.tlt14.paper.pdf.
[13] Jerry Fodor, Language, thought and compositionality, Mind & Language 16 (1) (2001), https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0017.00153.
[14] Mirjam Fried, Jan-Ola Östman, Construction grammar: a thumbnail sketch, in: Mirjam Fried, Jan-Ola Östman (Eds.), Construction Grammar in a Cross-Language Perspective, John Benjamins, 2004, pp. 11–86, https://www.researchgate.net/publication/280291354_Construction_Grammar_A_thumbnail_sketch.
[15] Ángel J. Gallego, Román Orús, Language design as information renormalization, https://arxiv.org/abs/1708.01525.
[16] Peter Gärdenfors, et al., Event boards as tools for holistic AI, in: 6th International Workshop on Artificial Intelligence and Cognition, 2019, pp. 1–10, http://ceur-ws.org/Vol-2418/paper1.pdf.
[17] Peter Gärdenfors, Frank Zenker, Theory change as dimensional change: conceptual spaces applied to the dynamics of empirical theories, Synthese 190 (6) (2013) 1039–1058, http://lup.lub.lu.se/record/1775234.
[18] Jose Emilio Labra Gayo, et al., Validating and describing linked data portals using shapes, https://arxiv.org/pdf/1701.08924.pdf.
[19] Joseph Goguen, An introduction to algebraic semiotics, with application to user interface design, in: C. Nehaniv (Ed.), Computation for Metaphors, Analogy, and Agents, Springer, 1999, pp. 242–291, https://cseweb.ucsd.edu/~goguen/pps/as.pdf.
[20] Suelen M. de Paula, Ricardo R. Gudwin, Evolving conceptual spaces for symbol grounding in language games, Biologically Inspired Cognitive Architectures 14 (2015) 73–85, https://www.sciencedirect.com/science/article/pii/S2212683X15000493.
[21] Helmar Gust, Carla Umbach, A qualitative similarity framework for the interpretation of natural language similarity expressions, in: Lucas Bechberger, et al. (Eds.), Concepts in Action – Representation, Learning, and Application, Springer, 2021, http://www.carla-umbach.de/publications/Gust&Umbach_QualitativeSimilarityFramework.pdf.
[22] Thomas Hartmann, Validation Framework for RDF-based Constraint Languages, Dissertation, Karlsruhe, 2016, https://publikationen.bibliothek.kit.edu/1000056458/3865145.
[23] Radek Kočí, et al., Object oriented Petri Nets – modelling techniques case study, in: Second UKSIM European Symposium on Computer Modeling and Simulation, 2008, https://ijssst.info/Vol-10/No-3/paper4.pdf.
[24] Michael Köhler, Heiko Rölke, Properties of object Petri Nets, in: International Conference on Application and Theory of Petri Nets, 2004, pp. 278–297, https://link.springer.com/chapter/10.1007/978-3-540-27793-4_16.
[25] Malka Rappaport Hovav, Beth Levin, The syntax-semantics interface, in: Shalom Lappin, Chris Fox (Eds.), The Handbook of Contemporary Semantic Theory, Wiley, 2015, pp. 593–624, Chapter 19, https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118882139.ch19.
[26] Corentin Jabot, Strongly-typed reflection on attributes, http://open-std.org/JTC1/SC22/WG21/docs/papers/2019/p1887r0.pdf.
[27] Wolfgang Jeltsch, Categorical semantics for functional reactive programming with temporal recursion and corecursion, https://arxiv.org/pdf/1406.2062.pdf.

[28] Karl Klose, Klaus Ostermann, A classification framework for pointcut languages in runtime monitoring, in: International Conference on Objects, Components, Models and Patterns, Proceedings, 2009, pp. 289–307, https://link.springer.com/chapter/10.1007/978-3-642-02571-6_17.

[29] Daniel Korenblum, et al., Managing biomedical image metadata for search and retrieval of similar images, Journal of Digital Imaging 24 (4) (2011) 739–748, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3138941.

[30] Yusuke Kubota, Robert Levine, The syntax-semantics interface of 'respective' predication: a unified analysis in hybrid type-logical categorial grammar, Natural Language and Linguistic Theory 34 (2016) 911–973, https://www.asc.ohio-state.edu/levine.1/publications/kl-resp.pdf.

[31] Jonathan Lawry, Yongchuan Tang, Uncertainty modelling for vague concepts: a prototype theory approach, Artificial Intelligence 173 (2009) 1539–1558, https://www.sciencedirect.com/science/article/pii/S0004370209000903.

[32] Martha Lewis, Jonathan Lawry, Hierarchical conceptual spaces for concept combination, Artificial Intelligence 237 (2016) 204–227, https://www.sciencedirect.com/science/article/pii/S0004370216300492.

[33] Yaoyong Li, Hamish Cunningham, Geometric and quantum methods for information retrieval, in: Association for Computing Machinery Special Interest Group on Information Retrieval, vol. 42(2), 2008, pp. 22–32, https://dl.acm.org/doi/10.1145/1480506.1480510.

[34] Luís Lopes, et al., A virtual machine for a process calculus, in: International Conference on Principles and Practice of Declarative Programming, 1999, pp. 244–260, http://di.fc.ul.pt/~vv/papers/lopes.silva.vasconcelos_virtual-machine-tyco.pdf.

[35] Fei Liu, et al., Coloured Petri Nets for multilevel, multiscale and multidimensional modelling of biological systems, Briefings in Bioinformatics 20 (3) (2019) 877–886, https://academic.oup.com/bib/article/20/3/877/4590142.

[36] Ilya Makarov, et al., Quantum logic and natural language processing, http://ceur-ws.org/Vol-1886/paper_16.pdf.

[37] Konstantinos Meichanetzidis, et al., Quantum natural language processing on near-term quantum computers, https://arxiv.org/abs/2005.04147.

[38] Laura A. Michaelis, Knud Lambrech, Toward a construction-based theory of language function: the case of nominal extraposition, Language 72 (2) (1996) 215–247, https://spot.colorado.edu/~michaeli/documents/Michaelis_Lambrecht_NE_LG.pdf.

[39] Friederike Moltmann, Parts and Wholes in Semantics, Oxford Univ. Press, 1997, https://www.researchgate.net/publication/244487318.

[40] Pattanasak Mongkolwat, et al., The national cancer informatics program (NCIP) annotation and image markup (AIM) foundation model, Journal of Digital Imaging 27 (2014) 692–701, https://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC4391072&blobtype=pdf.

[41] Lee J. O'Riordan, et al., A Hybrid Classical-Quantum Workflow for Natural Language Processing, Machine Learning: Science and Technology, vol. 2, 2021, https://iopscience.iop.org/article/10.1088/2632-2153/abbd2e/pdf.

[42] Paolo Pareti, et al., SHACL constraints with inference rules Paolo Pareti, in: C. Ghidini, et al. (Eds.), International Semantic Web Conference 2019, 2019, pp. 539–557, https://www.southampton.ac.uk/~gk1e17/shacl-inference.pdf.

[43] Walter B. Pedriali, The Routes of Sense: Thought, Semantic Underdeterminacy and Compositionality, Dissertation, St. Andrews 2011, https://research-repository.st-andrews.ac.uk/handle/10023/3142.

[44] Stefan Ramson, Robert Hirschfeld, Active expressions: basic building blocks for reactive programming, https://arxiv.org/abs/1703.10859.

[45] Martin Raubal, Formalizing conceptual spaces, http://www.raubal.ethz.ch/Courses/288MR_Spring08_Papers/Raubal_FormalizingConceptualSpaces_FOIS04.pdf.

[46] Daniel L. Rubin, Kaustubh Supekar, Annotation and image markup: accessing and interoperating with the semantic content in medical imaging, IEEE Intelligent Systems 24 (1) (2009) 57–65, https://web.stanford.edu/group/rubinlab/pubs/Rubin-IEEEIntelSys-2009.pdf.

[47] Vahid Salari, et al., Parametrized quantum circuits of synonymous sentences in quantum natural language processing, https://assets.researchsquare.com/files/rs-220713/v1_stamped.pdf.

[48] Christian Schafmeister, Alex Wood, Clasp Common Lisp implementation and optimization, in: 11th European Lisp Symposium, 2018, pp. 59–64, https://dl.acm.org/doi/10.5555/3323215.3323223.

[49] Roger Schaer, et al., Web-based tools for exploring the potential of quantitative imaging biomarkers in radiology: intensity and texture analysis on the ePAD platform, in: Adrien Depeursinge, et al. (Eds.), Biomedical Texture Analysis: Fundamentals, Tools, and Challenges, Academic Press, 2017, http://bigwww.epfl.ch/publications/schaer1701.pdf.

[50] Daniel D. Sleator, Davy Tamperley, Parsing English with a link grammar, https://www.link.cs.cmu.edu/link/ftp-site/link-grammar/LG-IWPT93.pdf.

[51] Gerold Schneider, A Linguistic Comparison of Constituency, Dependency and Link Grammar, Diploma, Zurich University, 2008, https://files.ifi.uzh.ch/cl/gschneid/papers/FINALSgeroldschneider-latl.pdf.

[52] Alexandru Telea, Jarke J. van Wijk, VISSION: an object oriented dataflow system for simulation and visualization, in: Data Visualization, Proceedings, 1999, pp. 225–234, https://pure.rug.nl/ws/portalfiles/portal/3178139/1999ProcVisSymTelea.pdf.

[53] Roman Urban, Megdalena Grzelinska, A potential theory approach to an algorithm of conceptual space partitioning, Cognitive Studies 1 (2017), https://ispan.waw.pl/journals/index.php/cs-ec/article/download/cs.1310/3065.

[54] Juan Uriagereka, Syntactic Anchors: On Semantic Structuring, Cambridge University Press, 2008, https://assets.cambridge.org/97805218/65326/frontmatter/9780521865326_frontmatter.pdf.

[55] Remi van Trijp, Cognitive vs. generative construction grammar: the case of coercion and argument structure, Cognitive Linguistics 26 (4) (2015) 613–632, https://www.degruyter.com/document/doi/10.1515/cog-2014-0074/html.

[56] Frank Zenker, From features via frames to spaces: modeling scientific conceptual change without incommensurability or aprioricity, in: Thomas Gamerschlag, et al. (Eds.), Frames and Concept Types: Applications in Language and Philosophy, Springer, 2014, pp. 69–89, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.901.7852&rep=rep1&type=pdf.

This page intentionally left blank

PART III

Conceptual spaces and graph-oriented data-modeling paradigms

This page intentionally left blank

CHAPTER

7

Multi-aspect modules and image annotation

OUTLINE

7.1 Introduction

Modularity is a canonical principle in software engineering: it is almost universally accepted that modularity is desirable, even if not practiced as ubiquitously as that would imply. In this chapter, we will present a version of this paradigm which we introduced in Chapter 4 as "multi-aspect" modular design. The key detail about this approach is that each module, in the general case, provides features associated with a range of software-development aspects, such as data persistence, GUI design, serialization, and capabilities related to runtime-reflection and participating in multi-component workflows. Multi-aspect modules embrace a compromise between monolithic stand-alone applications and the more narrowly focused modules typical of conventional modular design. Our discussion of modules' "aspects" builds off of ideas presented in earlier chapters, including the intuitive "Semiotic Saltire" schematic diagram, where database, serialization, GUI (or "visual objects"), and "code models" were pictured as four aspects circling around a central data type (mim-

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00014-3

icking the "saltire" pattern of, e.g., the Scottish flag).

7.1.1 Comments on procedural and database aspects

For our purposes, we will define a *multi-aspect module* (or just "module" in the present context) as a *code library* that, in typical cases, includes procedures addressing multiple *aspects* of software implementation. In particular, modules will have groups of procedures concerning data persistence/database integration, serialization, and GUI visualization. In addition, modules will typically have some notion of a "code model" and have capabilities for dynamically invoking procedures or functionality exposed by the module. For the sake of discussion, we will refer to modules' "code modeling" dimensions in this sense as "procedural exposure."

Before addressing image-annotations in particular, we will make some general comments about the four "aspects" we have focused on, namely serialization, GUIs, data persistence, and (what we are calling) procedural exposure.

The procedural-exposure aspect includes (at least) two distinct possibilities. First there is the goal of *runtime-reflection*, or calling a modules' procedures by describing the desired procedure and its parameters. In this scenario, the origin of requests to execute reflected procedures lies outside the module's own code. Moreover, these requests are not compiled into some application or some other module, but rather are dynamically generated; as a result, the relevant module in the procedure cannot be called directly, but rather a request to run that procedure must be encoded indirectly. If a module is implemented as a C++ code library, for example, then it is straightforward to implement procedures as externally accessible to other C++ modules, which can call the relevant procedure as an ordinary C++ function.

Choosing procedures to expose to sibling modules in the same language and environment as the original module is indeed one dimension of code modeling, but runtime reflection more generally involves a different scenario, where the procedure requests originate from a source that is *not* the same language as the original module, or where for some reason the procedure cannot be called directly. In these cases, the caller encodes a data structure describing the procedure and its parameters, and the module must supply capabilities to interpret such requests and route them to the correct procedures.

The second manifestation of *procedural exposure* is similar to runtime reflection, but in this case requests need to be mapped to specific procedures implemented within a module. Instead, modules may define different "services," "endpoints," or "request handlers"—following our precedent in Chapter 6, we will use the term *meta-procedures*—which are functionally similar to procedures (in the sense of having input and output channels and being executed as a self-contained process), but may not be directly implemented in a single function-body. Request handlers are characteristic of web services: web URLs often encode the name of an endpoint that is available through the corresponding web application, but the names of actual procedures defined in the computer code responsible for the web application is rarely exposed to the world wide web. A *meta-procedure* in this sense is similar to a service or API endpoint provided by a web application, except meta-procedures do not necessarily communicate or encode data according to web protocols (such as HTTP).

Some clarification may also be in order with respect to aspects involving data persistence. We assume that modules provide code that cuts across multiple aspects in a relatively self-contained manner; for instance, each module provides its own serialization and GUI code. This approach differs from conventional modular design, where more likely a single module will address only a single "aspect"—for instance, a component dedicated to data analytics would be less likely, according to convention, to include serialization and GUI classes.

Our notion of "multi-aspect" modules therefore assigns greater scope and autonomy to modules, almost as if each module is a kind of mini-application. However, modules are still intended to be pieced together with other modules to form applications proper. Multi-aspect modules are not intended to be *entirely* self-contained.

In the area of data persistence, it would be impractical for every module within an application to maintain its own separate database. We assume therefore that applications are designed with data-persistence capabilities that are external to and shared across most modules. Instead of communicating with databases directly, modules would then interface with some application-level "kernel," which in turn would abstract away low-level database interop details. One benefit of this arrangement is that said kernel can embrace data-representation strategies that transcend particular database architectures, translating such more generic structures into specific database contents on a case-by-case basis. This is one motivation for our model of "pre-persistent" representations, discussed in Chapter 6.

In Chapter 6, we also discussed virtual machines for query-evaluation, which are also applicable for working with pre-persistence representations in a multi-aspect modular context. Virtual machines can be engineered to operate directly with data structures formulated according to pre-persistent representations. The set of operations exposed by VMs in this context could likewise include features for marshaling information structured according to "pre-persistence" representations into formats endemic to specific database genres.

Having therefore clarified certain details specific to data persistence and "procedural aspects" of modules in general, we can present specific examples through the case of image-annotations, continuing the outline of annotation-related data formulated in earlier chapters.

7.1.2 Assessing the proper scope of an image-annotation module

In Chapter 4, we suggested limitations within (informal) "application-centric" workflows, where usage-patterns involving specific popular applications tend to become entrenched and relatively inflexible. We argued that these usage-patterns demonstrate a kind of inertia which inhibits the emergence of more flexible workflows, where components can be pieced together in a more open-ended fashion. Workflows are more flexible when their component parts tend to be more narrowly focused, such as code libraries, rather than monolithic applications. This is (at least in part) due to narrower components being explicitly designed with the anticipation of interoperating with other components to be useful, whereas standalone applications are built to serve users' needs on their own: interoperability with other components may be desired capabilities provided through plugins, extension, or "advanced" features, but they are less crucial to the applications' usability in the first place.

Software workflows accordingly focus on components that are designed to be used as one element within a larger computing environment. Similarly, modular design emphasizes the virtues of software systems being assembled from operationally isolated parts. Both workflow designs and modular software-development, then, are grounded in relatively streamlined components targeted at specific areas of functionality. However, defining the scope of components' desired features *too* restrictively also raises complications. There is therefore a tension between conceptualizing workflows and modular design too narrowly or too broadly: components that act like monolithic applications present the risk of "ecosystem fragmentation" and the failure to achieve modular design spaces that are flexible and open-ended with respect to how components are pieced together, but components that are too narrowly targeted to individual software-development concerns can re-

quire extra effort to be integrated into overarching projects, mitigating the potential benefits of modular design. Multi-aspect design attempts to engineer an optimal balance between these two extremes.

This chapter will take image-annotations as a use-case motivating the basic ideas of multi-aspect modular design. This chapter will also continue our analysis of image-annotation data, which we reviewed at the end of Chapter 6 in the context of the Annotation and Image Markup (AIM) format. Image annotation presents a good case study for data-integration problems in general, because properly interpreting annotation data requires integrating at least four different data models: annotation (geometric) descriptions themselves; visual presentation details (how annotations should be displayed on-screen); image-acquisition metadata (recording the provenance of image series, color depth, resolution scale, and other factors which determine how images' pixel data should be construed optically); and clinical/diagnostic background, which permit image-annotations to be employed as a form of biomarker. Because image-annotations are connected to each of these four separate domains (and possibly others, depending on how one wishes to demarcate domain-boundaries), bioimage-annotations are likely to play a role in many different diagnostic, research, and/or clinical-decision contexts, meaning that annotations will, in general, be synthesized with different forms of associated data (such as tissue biopsies, treatment plans, health records, and so forth). Annotations therefore offer a convenient window on the sorts of issues that arise when data structures with significantly different profiles need to be merged into a single information space (for purpose of analysis, machine learning, unified data persistence, interoperating GUI components, and so forth).

Image annotation and segmentation is an important analytic process in many scientific, technical, and commercial fields. Nonetheless, there are few standard formats for describing and representing image annotations, and these tend to be domain-specific.[1] This is not a new observation; Daniel L. Rubin *et al.*, in 2007, note that:

> Images contain implicit knowledge about anatomy and abnormal structure that is deduced by the viewer of the pixel data, but this knowledge is generally not recorded in a structured manner nor directly linked to the image[;] the *terminology* and *syntax* for describing images and what they contain varies, with no widely adopted standards, resulting in limited interoperability. The contents of medical images are most frequently described and stored in free-text in an unstructured manner, limiting the ability of computers to analyze and access this information. There are no standard terminologies specifically for describing medical image contents—the imaging observations, the anatomy, and the pathology. [N]o comprehensive standard appropriate to medical imaging has yet been developed. A final challenge for medical imaging is that the particular information one wants to describe and annotate in medical images depends on the *context*—different types of images can be obtained for different purposes, and the types of annotations that should be created (the "annotation requirements" for images) depends on that context. For example, in images of the abdomen of a cancer patient (the context is "cancer" and "abdominal region"), we would want annotations to describe the liver (an organ in the abdominal region), and if there is a cancer in the liver, then there should be a description of the margins of the cancer (the appearance of the cancer on the image). ***[21, pages 1–2]***

These challenges inspired the Annotation and Image Markup project, which "provides a solution to the ... imaging challenges [of]: No agreed upon syntax for annotation and markup; No agreed upon semantics to describe annotations; No standard format ... for annotations and markup."[2] However, AIM has been adopted

[1] Such as AIM for instance, or DICOM-SR (DICOM structured reporting), CVAT (computer vision annotation tool) (XML), COCO (common objects in context) (JSON), PASCAL VOC (pattern analysis, statistical modeling and computational learning visual object classes) (XML).

[2] https://wiki.nci.nih.gov/display/AIM/Annotation+and+Image+Markup+-+AIM.

most noticeably in cancer research and radiomics, less so in other biomedical areas.

One obstacle to formalizing image-annotation data is that annotations have a kind of intermediate status, neither intrinsic parts of an image nor merely visual cues supporting the presentation of the image within image-viewing software. Specifically, many applications allow markup or comments to be introduced with respect to an image; from such applications' point of view, annotations in this sense are part of the application display, not part of the image—analogous to editing comments that might be added to a text document by a word processor or PDF viewer, which are records of user actions, not intrinsic to the document itself. In DICOM (the "Digital Imaging and Communications in Medicine" format), for instance, "presentation state" (one mechanism for recording image annotations) includes all details about how the image currently appears to DICOM workstation users (radiologists, etc.), which may include markings such as text-annotated distinguished regions (*see*, e.g., [8] or [13]). Insofar as image-annotations are considered to be artifacts of image-viewing software, rather than significant data structures in their own right, there is less motivation for imaging applications to support canonical annotation standards.

Nevertheless, in many scientific and technical areas image annotations *are* significant; they are intrinsic to the scientific value of a given image as an object of research or observation. Image regions, segments, and features have a semantic meaning outside the contexts of the applications that are used to view the corresponding images, which is why it is important to develop cross-application standards for describing and affixing data to image annotations.

Though image analysis serves different goals in different contexts (e.g., segmentation of microscope images to detect cancer cells serves different ends than segmentation of street-level camera snapshots to study traffic patterns), there is always a possibility of analytic techniques developed in one subject area to be applicable elsewhere. Furthermore, certain computational domains are similar enough to image analysis to warrant inclusion in a general-purpose image-annotation framework, even if the underlying data does not contain "images" in the conventional sense (not, for instance, captured via photographs or microscopy). For example, PDF document views, flow cytometry (FCM) data plots, and geospatial maps subject to geographic information systems (GIS) annotations may all be considered images—by virtue of a semantic significance attributed to color and to geometric primitives as a way of characterizing phenomena observed or modeled through their data—even though such resources are not acquired by ordinary "image-producing" devices.[3]

Most applications skip defining "images" as such, and therefore do not consciously delineate the scope of image annotation overall.[4] Here, we consider imaging to be more general than just graphics obtained by a direct recording of the optics of some physical scene via cameras, microscopes, or telescopes. That is to say, the image acquisition process is not necessarily one where data is generated by an instrument that produces a digital artifact by absorbing light, so that geometric and chromatic properties of the image are wholly due to the functioning of the acquisition device. How broadly one *should* define imaging *per se* (which remains an open question) affects the range of domains whose semantics could reasonably be incorporated into annotation frameworks. For example, if immunofluorescent flow cytometry (FCM) data plots are classified as images, then the numerical properties

[3] *See* [24] for an interesting discussion related to the GIS case.
[4] The "Pantheon" project, characterized as a "platform dedicated to knowledge engineering for the development of image processing applications" (*see* https://hal.archives-ouvertes.fr/hal-00260065/document), offers one of the few attempts in imaging literature to rigorously define "imaging" and "image processing" in the first place. Pantheon (via its **Pandore** component) also includes an image processing *objectives* ontology.

of the "channel" axis, with notions of "decades" and a "log/linear" distinction, become relevant to the annotation vocabulary for representing spatial dimensions and magnitudes.[5]

After devoting the first part of this chapter to a relatively detailed examination of image-annotation data, the remainder of the chapter will turn attention to other bioinformatic areas. We will consider data-integration strategies in the context of modeling image-annotations (and image biomarkers) alongside other varieties of biomedical information.

7.2 Image annotations: Core data models

The first step when formulating a general-purpose image-annotation data model is to consider the underlying representation of image "points" or "locations" *ab initio*. As we will show next chapter in the context of AIM, this is a more intricate problem than it may appear at first glance.

An image annotation has the distinguishing characteristic of being a *geometric* object, but one whose meaning is only available *in conjunction with* an accompanying image (or 3D model, and so on for higher dimensions; we assume that the scope of image-annotations can be extended to include time points and intervals, and so be applied to 4D media as well). Any geometric entity, such as points, lines, or regions, therefore needs to be defined in relation to the accompanying image (sometimes called the *ground* image). For example, any magnitude in the annotation data needs to be evaluated relative to the size and resolution of that image. An intrinsic feature of any annotation-description is therefore the *scale of resolution* at which it applies to the ground image.

We might assume by default that annotations are always markings targeted at the ground image in its "internal" (100%) zoom level. In this case, it should be guaranteed that data about ground-image dimensions can be ascertained from whatever data structure or object represents the ground-image itself. For example, if annotation magnitudes are expressed in millimeters, the numbers are meaningful only after establishing the width and depth of the ground image in millimeters accordingly (assuming it is viewed at its "natural" scale, without zooming either in or out as an artifact of the visual display). Such image details must therefore either be an internal field or group of fields within the annotation data or else must be accessible through the object representing or referring to the ground image (here we will use the generic term "object" to mean any integrated data structure, not necessarily endowed with object-oriented designs). All annotations accordingly have an *intrinsic* scale, which represents the *internal* details of how magnitudes in the image relate to points/intervals/locations in the ground image. This intrinsic scale may be different from the visible scale through which annotations are presented: if the ground image is zoomed in or out, any displayed annotations would need to be scaled equivalently. The particular dimensions of an annotation *insofar as they are viewed* in a software window are therefore artifacts of the *rendering* of annotations, rather than manifestations of data that are *part* of the annotation.

A complicating factor, however, is how annotations are created in the first place. Suppose an annotation is marked on an image that the radiologist (say) views at 200% scale: the annotations visible to the radiologist are therefore double the size of their "intrinsic" data. Presumably any annotating software is capable of scaling the on-screen rendering to compensate for zoom effects. The *visible* rendering of the annotation is not the same thing as the annotation itself; so when a radiologist marks a one-centimeter length on a 200% zoomed image, the software would internally record the annotation

[5] *See* [27, e.g., page 18] or [26, especially pages 14*ff*], for example.

data as scaled so that the corresponding length is one half-centimeter. All this is straightforward. However, to precisely record the conditions under which an annotation is created—perhaps so as to re-evaluate diagnostic findings supported by the annotation—it may be appropriate to notate the fact that *from the perspective of the user* who created the annotation in the first place, they were viewing the annotation (and likewise the ground image) at double-scale. For each annotation, then, one *may* wish to record the resolution at which it was viewed when first created, if this in fact differs from the default ground-image scale.

Similarly, when an annotation is subsequently viewed, the software will typically support zoom operations, so the *state* of the annotation currently visible may be different than its "intrinsic" dimensions. It should be possible to bundle the annotation *together with the view state* as a larger data structure associated with any annotation *and also* to isolate the annotation from any particular view-context. View-state details (such as zoom factors) are not significant to how the annotation is interpreted or analyzed, and as such should not be confused with *intrinsic* data for purposes of data-sharing. On the other hand, in some contexts it *is* desired to share visible state: consider a telepathology scenario, where a doctor and a radiologist discuss an image from two different locations (in the sense of, say, two different cities). Presumably they should be able to synchronize their views so that they see the image and annotations at the same scale, with the same coloration, and so forth.

Considerations of view state also apply to details such as colors and line width. Ordinarily, colors are a presentation detail rather than intrinsic to annotation data; there is no geometric significance attached to a line being drawn as yellow, rather than red, for instance. Colors may be pressed into service as a classification device: one may want to distinguish annotations playing different interpretive roles. For example, circles or polylines outlining a region-of-interest might be classified as playing a different role than line segments whose purpose is to calculate some biologically significant length. In this sort of situation annotating software may use colors as visual cues to the corresponding role. However, only the roles themselves, not the colors that signify them, are "intrinsic" to the annotation. Colors *are* internal to view-state data, and should be modeled within this data, alongside details such as image/annotation zoom factors. Similar points apply to opacity (if annotations are semi-transparent to allow some of the ground image to remain visible) and to line-width (lines may be thickened to make them more visible, but most annotation data structures consider lines to be hypothetically infinitely thin extants, which serve merely to connect two points, establish a length, or form part of a polyline encosing a region). Similarly, line-styles (using dashed, dotted, or "wavy" lines, say, instead of straight ones) could be employed as a way of connoting roles or other non-geometric information (consider a system that renders annotations marked with less confidence via dashes, rather than solid lines).

With that said, however, it would be premature to rule out the possibility that certain kinds of annotations would make *intrinsic* use of features such as colors, line-styles, and opacity. As a result, a general-purpose annotation data model should identify information that is *always* intrinsic to the annotation itself (such as geometric vertices) as well as view-state details, which *usually* are presentational rather than intrinsic data; but should allow for some typically presentational data, in specific contexts, to be treated instead as intrinsic to the annotations.

7.2.1 Magnitudes and coordinates

Assuming, then, that we have accounted for ground-image details, such as the image's intrinsic dimensions, and have identified which

annotation data is intrinsic and which is presentational, the annotation itself will subsequently be embodied via geometric entities (such as points, lines, and regions) referring to locations and intervals in the ground image. Typically an annotation is a geometric structure whose meaning is dependent on some separate interpretive declaration: a point or line has no significance other than its geometric role in fixing the annotation's shape.[6]

As purely geometric entities, points thereby represent "locations" within the image, and line-segments represent intervals within the image's internal "space." This language is inexact, because it is subject to different interpretations. We could see image "locations" as individual *pixels*, which, among other consequences, has the effect that the building-blocks of locations are integers (there is no such thing, at least speaking literally, as, e.g., a "half-pixel"). Alternatively, locations can be treated as abstract (dimensionless) points in the "space" inexactly represented by an image. This choice conforms to the general distinction between *raster* and *vector* graphics, where the former is pixel-based and the latter is more mathematical. Raster and vector have distinct use-cases, so a general annotation model should support representations conducive to both kinds of graphics

One consequence of this generality is to preclude *a priori* judgments about how magnitudes and coordinates should be represented. Image locations may be understood as integers in some contexts and as floating-point values in others. Moreover, floating-point values can have different levels of precision in different contexts (including "infinite-precision," which lets quasi-real numbers extend across arbitrary amounts computer memory, for any desired degree of precision).

It is possible for an annotation system to allow two different numeric types, insofar as numbers designating locations have a different purpose than those marking quantities such as rotation angles or ratios/eccentricities; we might use integers or **float**s for point-locations, but **double**s for magnitudes that are not fixed to spatial points. Note also that "quasi-reals," by which we mean numbers intended to approximate the full real-valued number line, do not necessarily need to be encoded according to the familiar floating-point standards. There are variant number systems, such as John Gustafson's "posits," which offer alternatives that can perform better than normal floats in some contexts [16], [3]. All told, then, there are multiple number systems that could potentially supply magnitudes for annotation geometry.

As we will discuss below, number-system flexibility is typified by libraries such as CGAL (for computational geometry), more than frameworks targeted more specifically at annotations proper.[7] In CGAL, any assertion of magnitude is always dependent on the specific number system used to mathematically ground the geometric system in the first place [14, pages 14 and thereafter, roughly]. For example, annotation points representing image-locations by encoding distances from the image's left and bottom sides (or left and top, and so forth) are dependent on fixing a convention for whether numbers are expressed as integers, ratios, quasi-reals, or instances of some other number system. Therefore another data point *intrinsic* to annotations constitutes specifications of what

[6] If such extra-geometric data as colors are (in some context) treated as *intrinsic*, then geometric elements in the annotation play the additional role of providing sites for the corresponding extra-geometric indicators. For example, if line-color is intrinsically significant, then a line becomes an object that can be assigned a color, and so it plays the role of permitting the color-data to be expressed in the annotation, alongside its role in defining a shape. For the current discussion, however, we will not consider extra-geometric data along these lines; hence points and lines have no "meaning" within the annotation, other than to construct a shape-representation, which is interpreted relative to the ground image.

[7] Including AIM, which only recognizes double-precision floating-point magnitudes.

kind of numbers the annotation uses in the first place (integers, **doubles**, single-precision floats, width/hight percentages vis-à-vis the ground image, etc.)

As seen in CGAL, coordinate systems that are more exotic than Cartesian spaces (such as polars, or "homogeneous"/projective coordinates) can also be utilized in some imaging/graphics contexts. Other special-purpose coordinate systems may additionally come into play for higher-dimensional media; 4D annotations, say, could potentially be facilitated via quaternions (*see* [2] or [19], for example).

In short, a general-purpose annotation framework should allow sets of annotations to vary over both the nature of magnitudes (integers, floats, etc.) and the coordinate systems to which those magnitudes apply. Any description of a collection of interrelated annotations should therefore notate such numerical details as intrinsic qualities of the annotation-set, even if not of particular annotations. In general, coordinate details would be fixed for multiple annotations in a set and therefore not re-declared for each annotation.[8] This implies that there must be some separate data structure representing an annotation "set" with higher cardinality than each annotation singularly. Depending on context, the extent of such a set may be all annotations on a given image, or image series (in the clinical sense of multiple images acquired at the same time as part of one diagnostic procedure), or larger image-collections in a database, and so forth.

This discussion points to several questions: How should we demarcate the extent of an annotation-set when defining a data structure that would hold information that applies to all of the annotations, such as magnitude and coordinate stipulations? Should the annotation-specific data model allow *nested* annotation-sets, or labeling annotation-sets with different roles (single-image, image-series, etc.)? Should coordinate information be unequivocally fixed across an annotation-set, or should individual annotations vary some of this data, e.g., some annotations expressed in Cartesian coordinates and others in polar? One possibility is to divide annotation-sets into subsets wherein every annotation shares the same coordinate setup, so at this *subset* level all coordinate data is homogenized.

An entirely separate issue is fixing units of measurement for points, lengths, and locations, such as centimeters, inches, or PDF coordinate points. Units may also be designated taking pixels as minimal spatial units, so that orthogonal distances are treated as (integer) multiples of pixels' width and height; we can speak of a horizontal line segment being 12 pixels long, for instance. Quasi-real lengths, such as line segments (not perpendicular to the image sides), can be modeled as the hypotenuse of the corresponding right triangles whose shorter sides are orthogonal pixel-lengths, so that pixel-based units can be generalized to non-integer values. Scalable Vector Graphics (SVG) recognize 9 different length units (including percentage relative to the whole image). It is reasonable to assume that each of these units might potentially be preferred as the basis of annotations in different contexts (for reference, [22], [1], [15], [6] present useful overviews/extensions for SVG).

To fully specify an image location, then, we need to determine three different factors: the nature of the raw magnitudes involved (e.g., integers or floating-point); the coordinate system wherein points should be oriented (e.g., Cartesian coordinates, with axes related to the center or sides of the ground image, or polar coordinates); and the scale-units for lengths. Coordinate system involves orientation as well as choice of a particular mathematical system; for

[8] Although one can envision scenarios where different coordinate systems would be useful for different annotations, even in the same set; for example, mixing Cartesian and polar coordinates for different regions of a single image, or expressing some locations via percentages of the image's overall dimensions (rather than units such as inches or millimeters).

instance, some graphics environments would center Cartesian axes on the center of an image, others on the top-left corner so that increasing numbers correspond to movement down or leftward; others the bottom-left or bottom-right, and so forth.

In most cases, all of this foundational data will be declared for groups of annotations rather than individual ones. Within the scope of a single annotation, we can take for granted that a pair of numbers (say) unambiguously designates a point in the ground-image. However, this assumption is dependent on all the requisite background data being shared alongside annotation-specific data in any data-sharing scenario.

7.2.2 Procedural and modeling considerations

Assuming we thereby have enough infrastructure to rigorously designate individual points, annotation-shapes can then be built up as point-sets, subject to multiple interpretations. Point-sets may be used to describe curved objects as well as straight line segments (and collections thereof), but for now, consider just straight-line cases. There are numerous options that have to be resolved when anticipating the design and implementation of an image-annotation module (we use this case study as a concrete illustration of procedural/data-modeling overlap as discussed last chapter).

Assume that annotations' shape data contains two or more points that describe an open polygonal arc or a closed polyline. There are several details to be considered. One is orientation: it is reasonable to model polylines, at least those which span a convex region, as point-sets where points are ordered in either a clockwise or counter-clockwise direction. Should the data model restrict how points are ordered, so that points are automatically sorted within the annotations' point-set or, alternatively, constructions that would yield inconsistent ordering be rejected?

This question amounts to the following: suppose we have a point-set that can be shown geometrically to span a convex region when the points are arranged in a certain order. Should that ordering be taken to be canonical, such that a different ordering declared among the points is interpreted as an artifact in how the points are assembled, which can be corrected when finalizing the annotation data? Or should annotations allow for self-intersecting polylines?

In the former case, should each annotation be given the option of clockwise or counter-clockwise ordering (insofar as chiral orientation is significant in some contexts), or—for the sake of argument—should the points automatically be repositioned in clockwise manner? Moreover, how should these specifications be enforced? Should a clockwise-ordered point-set be considered identical to a second set with the same points, but a different order? Or should self-intersecting point-sets (assuming one simply creates line segments by following the points in the order given) be considered malformed? Also, how do we deal with anomalies such as multiple points being duplicated in the point-set? Should the extraneous points be eliminated, or should data with those characteristics be deemed malformed and unusable?

Another line of questions concerns closed polylines versus open polyarcs. How should we notate the difference? Should we require closed polylines to start and end with the same point? Note that if so we need two different identical points to be part of each point-set, so point-set ordering becomes consequential. Alternatively, we could model polylines and polyarcs as two different shape *types*, where the former (but not the latter) have an implicit edge between their last and first points (again this requires that ordering be fixed). Another possibility is to introduce a "pseudo-coordinate" such as **TikZ**'s "**cycle**," which yields a point duplicating the first in a sequence (how to accommodate those

special coordinates, needing contextual interpretation, in a point data-type then becomes a concern for implementations). Finally, a fourth option is to use some sort of flag to denote the intention to treat a given point-set as spanning a closed path, rather than open arc (this is potentially more flexible, because it does not foreclose options in how to deal with point ordering). This last approach is taken in this book's demo code.

Another consideration for representing polylines and polyarcs is how to deal with (three or more) colinear points. Technically, a point midway between two other points in the same point-set, where all three lie on the same line, is extraneous. Should points in such circumstances simply be dropped from the annotation, or should constructions involving colinear points be rejected as malformed? This question depends on whether we consider intermediate colinear points to have any semantic value or meaning (despite their apparent mathematical superfluousness).

Consider the following scenario: A user creates and then edits an annotation by dragging handles representing vertices in the annotation-shape. The user might try to visually form the annotation so as to encircle a region-of-interest, let's say. This leaves open the possibility that they may drag a vertex to a point colinear with two adjacent vertices. However, it would not make sense to simply eliminate that vertex, because the user may potentially drag it once again, which could result in the colinearity being broken. Since the annotation might be stored and edited again at some future time (potentially by a different person), one can argue that it is premature to simply drop intermediate colinear points from the point-set. On the other hand, it may be appropriate to *flag* them as superfluous from a mathematical point of view with respect to the annotation's current state.

This modeling decision translates to procedural functionality: if extraneous colinear points are permitted to be part of an annotation's point-set, but excluded from geometric algorithms, a natural operation to "restrict" the annotation to its geometrically valid point-set would be to offer a procedure that maps the point-set onto a filtered version with no superfluous points. One could similarly implement "normalizing" functions which address concerns such as point-order. In other words, even if the annotation's data model does not explicitly stipulate restrictions or policies for (say) self-crossing point orders, code libraries instantiating the data model could provide procedures to convert more free-form annotations to ones which obey stricter geometric guarantees.

Note in particular that many issues we have described here as *data modeling* questions similarly have procedural ramifications. The issue of colinearity organically gives rise to the notion of normalization procedures that filter out intermediate colinear points. Issues of chirality and self-intersection suggest procedures to order point-sets into clockwise or counter-clockwise sequences (and to invert them so as to switch orientations). Both colinearity and chirality/convexivity come into play with respect to validating point-set representations and/or building point-sets incrementally: What procedures are implemented to add a new point to an existing set, or to initialize a point-set from multiple points *ab initio*? What are those procedures' pre- and post-conditions? For example, should a precondition for initializing a polyline be that the supplied points span a convex region? These are examples of how data-modeling decisions tend to translate to coding requirements, the pattern of intersection between data and code which we discussed earlier.

7.2.3 Annotations with curved geometries or cross-references

The last few paragraphs have focused on annotation-shapes based on straight line segments. Of course, many annotations are better expressed via curved paths, whether open or closed. Some curves can be specified via small

point-sets; e.g., ellipses may be defined by asserting their focal points and one point on the curve itself. Such representation tactics, relying solely on point-sets to construct shapes, are predominantly employed by AIM, for example. In general, though, curve-definitions require combining points with scalar magnitudes representing distances, rather than locations. Circles have centers and radii; ellipses have focal points and eccentricities. Circular or elliptical *arcs* can be defined by constructing the full circle/ellipse and then providing start/end angles.

The point here is that annotation data structure may incorporate sets of magnitudes (call them "length-sets") as well as point-sets. Sometimes length-sets can have additional structure; for example, one common strategy for defining ellipsoids (generalizing from 2D ellipses to higher dimensions) is via "covariance matrices," paired with designation of a single (center) point. In that case, the lengths are associated with matrix row/column positions. Length sets might potentially be used for polylines as well as curves: for a regular polygon with n sides and d diagonal, it is probably more convenient to assert the shape's center as a point and the remaining data as lengths (calling n a "length" just in the sense that it is a scalar quantity rather than the location of a point).

Consider also *arrows*, which are often employed as image overlays. Arrows can be a visual device to call attention to a specific image location or region. If arrows are recognized as a distinct annotation-type, then at a minimum one should identify a point which the arrow "targets." Alternatively, we could allow arrows to be paired with other annotations, so that the arrow "points at" their shape. Arrows may also connect two different prior annotations. Factors such as arrows' length and width, or the styling of the arrow "head," may be either visual/presentation details or intrinsic to the arrow-annotation. Assuming at least some data pertaining to the arrows' shape is deemed intrinsic, then this would be another case where length-sets (e.g., arrow length and width) would be needed as part of the shape data. Arrows also illustrate the condition that some annotations may reference *other* annotations, representing additional data fields that are not subsumed under point-sets or length-sets.

Curves that are more complex than circles or ellipses may also be constructed via techniques such as **b-splines** (which use point-sets as "control points" deforming the shape into the desired curve form) or by stipulating a particular genre of curve (e.g., parabolic or sinusoidal). Conceivably, one could introduce a generic "open-curve" shape-type, one which includes a data field labeling the more specific kind of curve intended.[9] These labels would then signal to the rendering engine that a particular algorithm should be used to generate the curve given the point and length sets included in the annotation data. In this context, geometric data such as bounding boxes, convex hull, deformation measures (how much a shape's area deviates from the smallest circle containing it), calculations as to whether a line segment between two points would intersect the curve, and so forth, would

[9]The distinction between curves and linear annotations with relatively simple geometries (such as polygons, polylines, ellipses, and arrows) and more complex shapes (requiring finer mathematical descriptions) is not only consequential for asserting annotations qua data objects, but also for graphical rendering. Depending on the graphics engine employed by whatever software displays annotations (against their target-image background), geometries with different levels of complexity may require different GUI treatment. For example, the Qt Graphics Scene classes support polygons and ellipses as graphics items that can be added to Graphics Scenes directly; also the QGraphicsPathItem path supports QPainterPaths which have some b-spline-related features. On the other hand, more complex curves (such as b-splines with some time-based or color variance, e.g. gradients or animations, that are tracked to the curve's "quasi-temporal" parameter) require custom paint events, and therefore intercepting the normal paint-event handlers for QGraphicsViews. If this duality in process-stages is handled seamlessly then shapes of arbitrary complexity can be treated as if they were ordinary QGraphicsScene items (see this book's demo code for some details).

need to be provisioned via algorithms specific to the curve's genre. In this situation the annotation framework might specify the *kinds* of algorithms that have to be available for arbitrary curve-types, but leave their implementations open-ended.

Summarizing our discussion so far, then, describing annotation-shapes in general requires a number of different data structures which would be intrinsic to annotation data: point-sets; length-sets, or in general collections of magnitudes which might supplement point-sets to uniquely fix a shape; in some cases, designation of *other* annotations referenced by a given annotation (such as an arrow, or, potentially, say, a circle grouping two or more other annotations); and, in some cases, designation of a special-curve genre with support for adding special-purpose rendering algorithms in a modular fashion. As this overview suggests, there are multiple data-structures that must be representable for general-purpose annotations even when defining the underlying annotation shape, setting aside other information (such as text description and image references), typically placed and/or asserted alongside annotation geometry.[10]

[10] As will be demonstrated in this book's accompanying code, one way to accommodate multiple annotation models is via some form of "state machine" employed when processing annotation data. The point in this context is that the nature of data provided (e.g., how many points/lengths are asserted for a given annotation) can signal which format is being used, and the proper resolving of data structures to intended formats can be clarified, or implemented unambiguously, via state machines. Because state machines in turn can be built atop Virtual Machines, there is potential for this particular annotation-processing mechanism to be exposed amongst operations in a relevant Virtual Machine, providing one further example of VM technology being applicable to (not necessarily self-evident) contexts. In sum, converting image-annotations between different data formats can be exposed and integrated within query-processing VMs (which of course would also be applicable to the goals of querying image corpora via properties of their annotations).

7.3 Annotations and image features

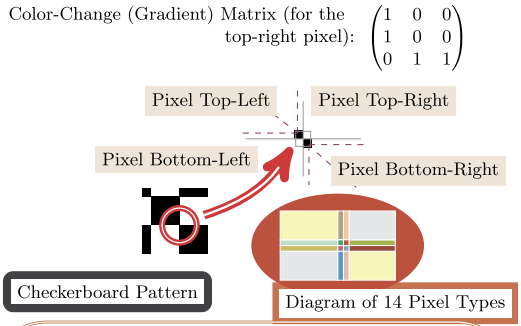
The previous discussion focused on annotations that have a fixed geometric outline, but what about descriptions or markup that extracts information from the image in other ways? For example, a point-set could be employed as a summarial overview of an image without the points being intended as vertices of a convex polygon. Consider an image-processing pipeline to count the number of cells in a whole slide imaging (WSI) view: one could match each cell with a distinct identifying point, then count the number of those points as proxies for the cells themselves. The point-set thereby summarizes the image without describing a particular connected shape. Point-sets could also potentially be used to model textures, diffusion processes, or other visual effects apparent in an image.

As image-processing has become more sophisticated, biomarkers extracted from an image have taken on a wider variety of forms, not only effects such as tumors or lesions, which can be circumscribed via a closed annotation-shape. For example, image analysis of cancerous tissues or nodules can reveal optical patterns that are indicative of tumor features such as heterogeneity, hypoxia (lack of oxygen), and angiogenesis (proliferation of blood vessels supporting the tumor). Such characteristics affect tumors' malignancy and aggressiveness: in general, hypoxic and heterogeneous tumors are more dangerous and resistant to clinical therapies.

Identifying these signals is a different process than, for instance, segmenting a tumor from a background of ordinary tissue; instead, suggestive indications for conditions such as heterogeneity, hypoxia, or angiogenesis involve patterns that can be mathematically extracted from extended regions in an image. For example, [5] document techniques for estimating heterogeneity by partitioning a nodule (as recorded via 2D graphics) into textural segments, where different sectors are isolated by considering the local

resemblance of neighborhoods around individual image-points to canonical patterns, such as "harmonic wavelets" (page 4). Similarly, angiogenesis (correlated with tumors' proclivity to expand and co-opt surrounding tissues) can be measured by detecting patterns in nascent blood vessels, which in turn are extracted (in mathematical image processing) by looking for point-neighborhoods that reflect the specific qualities of vascular nodes, where a tree-like pattern splits from larger branches to smaller ones (*see* e.g. [9] and [12], or [25], although the latter documents methods oriented more towards morphology and edge-detection than in the above overview).

In general, these kinds of textural analyses are similar to image segmentation or edge/contour-detection based on colors—where segments are assumed in general to be regions of similar color that are bounded by adjacent (background) regions with different colors; the boundary between a segment and its background is therefore defined by a noticeable displacement in color space—except that the basic analytic units are "textures," rather than colors. Regions of similar colors are quantified by measuring color-distances between different points, which is straightforward because color space is easily metrized into HSV (hue, saturation, value) or RGB (red, green, blue) coordinates. Quantifying textures, however, is more difficult, because texture-data is not manifest directly within individual pixels the way colors are. However, it is possible to mathematically calculate the degree to which the neighborhood of a given pixel approximates what would be expected if the image perfectly matched a particular texture around that point (Fig. 7.1 is a very simple depiction of this sort of analysis). Texture can therefore give rise to vectors of "signals" mapping points to one or more magnitudes characterizing the textural pattern around each point, to the degree that this pattern is approximated by the image data. These texture-vectors can then take the place of color vectors (i.e., triples such as HSV values) for segmentation, edge-detection, and similar image-analysis operations.



All pixels in this image can be sorted into 14 groups: white pixels surrounded entirely by white; black pixels surrounded by black; white (respectively black) pixels on horizontal or vertical edges between black and white edges; and pixels at black/white corners. Each of these groups can be associated with a distinct gradient matrix showing jumps in color values along eight directions relative to each individual pixel (the diagram above shows some of these groups by assigning them distinct colors, though the actual pixels are only black/white).

For real-life images, checkerboard patterns would presumably only be exact, so the gradient matrices would have entries other than perfect 1s and 0s. However, each matrix could still be clustered into one of the 14 canonical matrices for the pixel groups, allowing the pixels to be classified, and same-group pixels should be arranged according to a pattern matching that diagrammed above. Patterns of groups in the neighborhood of individual points can similarly be gathered into matrices, e.g.:

$$\begin{bmatrix} Group1 & Group2 & Group3 \\ Group1 & (center) & Group3 \\ Group1 & Group2 & Group3 \end{bmatrix}$$

The group-number of the center for each such matrix would predict the surrounding groups, and a secondary matrix could check whether these predictions are accurate (using "1" to mean a correct group, and "0" an unexpected one):

$$\begin{Bmatrix} 1 & 1 & 1 \\ 1 & (center) & 1 \\ 1 & 1 & 1 \end{Bmatrix}$$

The closer these matrices are to having all 1s, adding over all points in the image, measures how closely the image approximates a perfect checkerboard.

This may not be the most efficient algorithm to perform such an analysis; we describe it here simply as an illustration of techniques used for detecting more complex textures.

FIGURE 7.1 Checkerboard pattern as a (very simple) texture.

Marking the presence of textures (or the fact of an algorithm having identified them), however, makes annotations more complex than regions or patches identified in terms of color alone, because to be thorough the annotation would not only have to identify the relevant region, but to describe the texture itself. Unlike colors, which can be specified with only three quantities, textures may involve angles, displacements, gradients, and other mathematical parameters. Such parameters can be associated with individual pixels to create image-like data structures within an image-processing pipeline, where pixel-data associating pixels with *colors* is replaced by data involving texture-definitions.

Image-analysis workflows may therefore have several intermediate processing steps defined by midstage images or image-like resources derived from earlier images in the workflow via feature analysis or morphological operators. As such, image processing operations do not always act directly on images themselves; sometimes algorithms are based instead on mathematical complexes derived from the image, but with their own quantitative properties. For instance, color-valued pixels may be replaced by matrices measuring the gradient of some image-feature field in eight directions around each point (an example would be "Sobel kernels" applied to the image intensity function [11, for instance]). Data structures for describing textures are therefore more complex than for colors, and (more to the point) it is harder to stipulate *a priori* specific forms these structures would take on. Though we can fully capture all digitally reproduceable colors within spaces like HSV, one cannot develop an exhaustive list of all sorts of textures that might be analytically relevant for image-processing; thus the details of texture data would need to be open-ended for a general-purpose annotation model.

7.3.1 Specifying annotations' roles and origins

A further detail that should be clarified is that of how image-annotations originate. Sometimes, of course, annotations are manually introduced on images by human users of image-viewing software. On the other hand, automated image segmentation—or similar algorithmic or AI-driven image processing without human intervention—yields partitions of images into regions, or identification of semantically important locations in an image, therefore generating annotations computationally. In short, descriptions should support both human-generated and computer-generated annotations.

These descriptive mechanisms become more complicated, however, when we consider the full range of computerized image-analysis capabilities, such as texture-analysis and similar feature-extraction techniques, which can potentially yield more complex data structures that represent statistical patterns evident in the image (often after mathematical transformations of the underlying image data). Image processing may yield analyses that overlap with annotation objectives, but may not intrinsically produce annotations in the conventional sense. For example, an algorithm to infer the number of nuclei in a cell-scale picture—or (not a biomedical example but useful) assess traffic patterns by counting cars—may rely on statistical analysis of some quantitative image feature (such as "zero-crossings") without in fact producing determinate image segments. As mentioned above, notating textural patterns and feature-vectors along these lines is more complex than simple polygonal or elliptical annotation-shapes.

For a given "semantic" task—that is, an image-processing objective whose end-result is not just image-related data but some empirical observation—image segmentation, or other analyses yielding annotations, are a means to an end: one *way* to count nuclei is to delineate the edges of distinct nuclei in distinct segments, and then

count the number of segments which result. However, statistical image-analysis may produce largely accurate results for such semantic tasks, given large image corpora (e.g., estimating traffic flows from highway cameras), without yielding artifacts such as human-visible segment representations. Or, in a different domain, AI-powered analysis of FCS (Flow Cytometry Standard) data could establish a largely accurate count of "events" (i.e., discrete FCS measurements of light-scattering and/or fluorescent properties of cellular-scale entities) without manual "gating" (referring to the conventional practice of scientists using geometric annotations of FCS data-plots to isolate and thereby count different event-types). An AI-powered analysis of image features, or likewise of flow cytometry (FCM) data, may yield calculations similar to those which for *human* users are achieved via image segmentation, manual gating, and similar operations that clearly yield annotation data.

The complication arises when AI workflows along these lines do not themselves yield results that would normally be considered annotations, but rather yield the desired empirical results for which the annotations would be a preliminary step, e.g., an approximate count of the number of nuclei in a slide-image or cars in a highway photo, without a precise segmentation of the image marking their respective borders.

An image annotation is, among other things, a visual (or viewable) record of some image-processing activity. If a radiologist manually clarifies a report to the effect that a given CT scan shows a tumor by circling the area where the tumor is visible, he or she is using the image annotation to communicate to others the thought-process that motivated the diagnostic conclusion. This is different than an AI-driven processor, which would automatically demarcate an image segment outlining the tumor and use geometric properties of that segment to derive a pathological finding. In short, the data conveyed in an annotation—an image segment, rendered precisely, or rendered indirectly via a circle or polygon around the segment—may be *intrinsic* to an image-processing operation: it may be data acquired *at one stage* in an analytic workflow. However, annotations may also be *retroactive*; if a radiologist circles a tumor, he or she has completed (at least mentally) the image analysis, and is using the analysis to summarize what occurred in the course of the analysis. Therefore any image-processing task can be associated with *ex post facto* annotations that summarize the process, even if they are not intrinsic to it.

To continue the example of counting nuclei from a microscopy image (or cars from a traffic camera), an AI-powered observation might be retroactively *justified* by providing a segmentation, where the number of cell-segments (likewise car-segments) matches the AI count. In lieu of precise segments, however, it may be simpler to provide location-points for the "geometric center" of each cell (respectively, car), or the points furthest apart in the direction of each car's front-to-back; these may be the statistical signals used for the car-counting process (such orientation-based enumeration makes more sense in the traffic example than the microscopy). Analogously, facial recognition does not need to rely on segmenting out regions (eyes, nose, lips), but rather can be based on distances between individual points (such as the inner corners of each eye).[11]

In any case, depending on the analytic algorithm used, it is often possible to identify some spatial/geometric feature or object that can be visualized in the image context, and which summarizes or legitimizes the analytic operation. This summarial data, then, can provide *retrospective* annotations, which allow human viewers to understand and review the algorithmic process. In short, simply because image-processing tasks may not generate annotation data as part of their internal activity, it is still possible (and

[11] *See*, e.g., [20], [17], [7], [10], or [18].

may be desirable) for the software operationalizing these tasks to implement annotation generators, where the resulting annotations document the operations for the scientific record and/or summarize them in GUI objects for the benefit of human viewers.

In general, then, we can distinguish human-generated from computer-generated annotations, and moreover leave open the possibility that some computer-generated annotations are *retrospective*: that instead of being internal to an imaging computation, they are indirectly produced subsequent to such a computation, for purposes of documentation and validation. Annotations that are not *retrospective* could be called *internal*, as in, internal to a given image-processing workflow (note that these are our proposed terms; they are not common meta-annotation vocabulary).

Related to the distinction between *internal* and *retrospective* annotations, we can also recognize a contrast between "immersed" and "descriptive" annotation (again these are idiosyncratic terms, but they seem appropriate for the contexts involves). An example of an *immersed* annotation might be an image segment, where calculating the boundary of the segment is intrinsic to a specific image-processing objective, whereas a *descriptive* annotation might be an arrow *pointing toward* that segment. Here the descriptive annotation is introduced primarily for the benefit of human viewers.

The immersive/descriptive distinction is not always clear-cut. Consider the following two cases: In one scenario, an image-segmentation routine precisely delineates a region of interest (e.g., an outline of a red bird against blue sky), notating the segmentation result via a two-color-depth transform of the original image. Image-viewing software then shows the segment indirectly by encircling it, producing, in effect, a secondary annotation intended to call attention to the primary (segment) annotation. Here, clearly, the segment itself (encoding by a separate two-toned image) is intrinsic to the original analysis, whereas the secondary annotation has a purely descriptive purpose.

However, consider an alternate scenario, where the original segmentation is done with less precision (this may be the case where there is a more muted color differential between foreground and background). Imperfect segmentation may still be adequate for some semantic task (e.g., identifying a bird's species). An analysis could obtain a rough segmentation by marking certain points highlighted by an edge-detector, then protruding the convex hull of the region outward so as to be sure of encompassing the whole bird-segment within a polygon, albeit allowing some background pixels into the polygon as well.

This approximate segment is only an indirect representation of the region of interest (which would be the avian sub-image clearly outlined with no background included), but for analytic purposes the rough polygon might be a reasonable substitute for the finer-grained segment, analogous to how a cubic or quartic polynomial may be an adequate approximation to a more complex curve. Depending on how it is used, the approximate segment may therefore be considered merely a visual cue connoting the region-of-interest, or a significant region in its own right. Such a distinction would, most likely, depend on whether the approximation is used itself as a basis for further analysis, or instead is mostly a presentation device. This usage-context would therefore indicate whether an "imprecise" annotation should be classified as *immersed* or *descriptive*.

All told, then, annotations may be *internal* or *retrospective*, and *immersed* or *descriptive*; *computer-generated* or *human-generated*; and *manual* or *automated*. These distinctions are independent of one another; retrospective annotations, for instance, could be either immersed or descriptive, and either computer-generated or human-generated. These aspects of annotations and/or groups of annotations can be notated in various ways. One option (adopted by the

book's demo code) introduces an *originator* object, which provides information about how annotations and/or annotation-groups were created. Such objects' data fields can then clarify the roles and mechanisms driving their correlated annotations.

When applicable, annotations' roles are also, for obvious reasons, closely implicated with calculations *about* an image performed with the aid of annotations. As is concretely demonstrated in the context of AIM, annotation data may include quantities such as the length of a line segment or the area of a region. More complex forms of image-features engender more complex calculations; consider fractal dimension estimation in the context of angiogenesis [4, page 4], [23, page 6], or Gradient Vector Flow (GVF) (with applications to image-segmentation in contexts such as diagnostic pathology/oncology) [29, page 85], [28]. In AIM, calculations can be presented via both text descriptions and formulae composed in the Mathematical Markup Language (MathML), but these cannot provide a detailed and machine-readable representation of data structures that may be used as generated algorithmically by computationally intensive feature-extraction methods. That further level of detail leaves open the question of how to properly encode the broad range of calculations that could potentially be applied within the context of an annotation.

In effect, a general-purpose annotation framework needs to consider how much detail should be represented as one shifts focus from annotations themselves to the segmentation methods, textural features, and biological interpretations that come into play to the degree that *image* details are taken as indicators or warrants for image *biomarkers*. We will consider this question in the next chapter.

References

[1] Greg J. Badros, et al., A constraint extension to scalable vector graphics, in: Proceedings of the 2001 International Conference on the World Wide Web, 2001, pp. 489–498, https://constraints.cs.washington.edu/web/csvg-www10.pdf.

[2] Enrique Martinez Berti, et al., Dual quaternions as constraints in 4D-DPM models for pose estimation, Sensors 17 (2017), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5579524.

[3] John Besseling, Anders Renström, A comparative study of IEEE 754 32-bit float and posit 32-bit floating point format on precision, https://www.diva-portal.org/smash/get/diva2:1463840/FULLTEXT01.pdf.

[4] Marie-Laure Boizeau, et al., Automated image analysis of in vitro angiogenesis assay, Journal of Laboratory Automation 18 (2013) 411–415, https://journals.sagepub.com/doi/pdf/10.1177/2211068213495204.

[5] Dmitry Cherezov, et al., Revealing tumor habitats from texture heterogeneity analysis for classification of lung cancer malignancy and aggressiveness, Scientific Reports 9 (2019), https://www.nature.com/articles/s41598-019-38831-0.

[6] Alexandre Carlier, et al., DeepSVG: a hierarchical generative network for vector graphics animation, in: 34th Conference on Neural Information Processing Systems, Proceedings, 2021, https://proceedings.neurips.cc//paper/2020/file/bcf9d6bd14a2095866ce8c950b702341-Paper.pdf.

[7] Wei-Lun Chao, Face recognition, http://disp.ee.ntu.edu.tw/~pujols/Face%20Recognition-survey.pdf.

[8] Engin Dikici, et al., Integrating AI into radiology workflow: levels of research, production, and feedback maturity, Journal of Medical Imaging (Bellingham) 7 (1) (2020), https://pubmed.ncbi.nlm.nih.gov/32064302.

[9] Charalampos N. Doukas, et al., Automated angiogenesis quantification through advanced image processing techniques, https://pubmed.ncbi.nlm.nih.gov/17946107.

[10] Yueqi Duan, et al., Topology preserving graph matching for partial face recognition, in: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), 2017, http://ivg.au.tsinghua.edu.cn/people/Yueqi_Duan/ICME17_Topology%20Preserving%20Graph%20Matching%20for%20Partial%20Face%20Recognition.pdf.

[11] Panteha Eftekhar, Comparative Study of Edge Detection Algorithms, Thesis, California State University Northridge, 2019, https://scholarworks.calstate.edu/downloads/ff365796m.

[12] Josef Ehling, et al., Micro-CT imaging of tumor angiogenesis: quantitative measures describing micromorphology and vascularization, Biophysical Imaging and Computational Biology 184 (2) (2014) 431–441, https://www.sciencedirect.com/science/article/pii/S0002944013007268.

[13] Marco Eichelberg, et al., Consistency of softcopy and hardcopy: preliminary experiences with the new

DICOM extensions for image display, in: The International Society for Optical Engineering, Proceedings, 2000, https://ui.adsabs.harvard.edu/abs/2000SPIE.3980...97E/abstract.

[14] Andreas Fabri, et al., On the Design of CGAL, the Computational Geometry Algorithms Library, Max Planck Institute, 1998, https://core.ac.uk/download/pdf/210674597.pdf.

[15] Georg Fuchs, et al., Progressive imagery with scalable vector graphics, in: SPIE 7881, Multimedia on Mobile Devices, Proceedings, 2011, https://vcg.informatik.uni-rostock.de/~schumann/papers/2010+/Rosenbaum-EI11b.pdf.

[16] John Gustafson, Isaac Yonemoto, Beating floating point at its own game: posit arithmetic, http://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf.

[17] Gary B. Huang, et al., Towards unconstrained face recognition, in: 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, Proceedings, 2008, http://vis-www.cs.umass.edu/papers/unconstrained_face_workshop.pdf.

[18] Patchaiah Kalaiselvi, Sivasamy Nithya, Face recognition system under varying lighting conditions, IOSR Journal of Computer Engineering 14 (3) (Sep.-Oct. 2013) 79–88, http://www.iosrjournals.org/iosr-jce/papers/Vol14-issue3/M01437988.pdf.

[19] Yang Li, et al., 4DComplete: non-rigid motion estimation beyond the observable surface, https://arxiv.org/abs/2105.01905.

[20] Feng Lu, et al., Adaptive linear regression for appearance-based gaze estimation, IEEE Transactions on Pattern Analysis and Machine Intelligence 13 (10) (2004) 2033–2046, https://www.ncbi.nlm.nih.gov/pubmed/26352633.

[21] Daniel L. Rubin, et al., Medical imaging on the semantic web: annotation and image markup, http://cedarweb.vsp.ucar.edu/wiki/images/d/d9/R_19.pdf.

[22] Michail Schwab, et al., Scalable scalable vector graphics: automatic translation of interactive SVGs to a multithread VDOM for fast rendering, Micromachines 11 (7) (2020), https://ieeexplore.ieee.org/document/9354592.

[23] Matvey Sprindzuk, et al., Computer-aided image processing of angiogenic histological samples in ovarian cancer, Journal of Clinical Medicine Research (2009), https://core.ac.uk/download/pdf/8701857.pdf.

[24] Agata Ciołkosz-Styk, Adam Styk, Measuring maps graphical density via digital image processing method on the example of city maps, Geoinformation Issues 3 (1) (2012) 61–76, http://bc.igik.edu.pl/Content/249/PDF/GI_2012_6.pdf.

[25] Ioannis Valavanis, et al., A novel image analysis methodology for the evaluation of angiogenesis in matrigel assays and screening of angiogenesis-modulating compounds, https://link.springer.com/content/pdf/10.1007%2F978-3-319-23868-5_5.pdf.

[26] Lili Wang, Robert A. Hoffman, et al., Standardization, calibration, and control in flow cytometry, Current Protocols in Cytometry 79 (2017), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=921423.

[27] Meredith Weglarz, et al., Evaluation of Voltration Approaches for Optimal Data Acquisition in Flow Cytometry, Thesis, Harvard University, 2018, https://dash.harvard.edu/bitstream/handle/1/37945126/WEGLARZ-DOCUMENT-2018.pdf.

[28] Chenyang Xu, Jerry L. Prince, Snakes, shapes, and gradient vector flow, IEEE Transactions on Image Processing 7 (3) (1998) 359–369, https://pubmed.ncbi.nlm.nih.gov/18276256/.

[29] Lin Yang, David J. Foran, Deformable models and their application in segmentation of imaged pathology specimens, in: Jasjit S. Suri, Aly A. Farag (Eds.), Deformable Models, Springer, 2007, pp. 75–94, https://link.springer.com/chapter/10.1007/978-0-387-68343-0_3.

This page intentionally left blank

CHAPTER

8

Image annotation as a multi-aspect case study

OUTLINE

8.1 Introduction

The previous chapter indicated many questions that should be answered by a comprehensive image-annotation framework. This does not mean that frameworks should enforce some choices at the expense of others, but they should at least identify details where data models may diverge, and encourage specific models and implementations to document their individual policies. In other words, a framework does not need to be a definitive implementation; it could also be a protocol identifying details that particular implementations should take into consideration.

8.1.1 Design questions for image-annotation modules

For the sake of discussion, we will reiterate in summarial fashion a number of the questions that were indicated in the last chapter.

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00015-5

1. What number systems should be recognized for defining magnitudes, so that basic quantitative data such as distance between an image point and the image sides (allowing image-locations to be identified) can be notated? How many different number systems (integers, floating-point values with different degrees of precision, non-standard quasi-reals) should be recognized?
2. Should an annotation system allow multiple number systems to co-exist, e.g., an integer scale for image points, but a floating-point scale for ratios, angles, and other calculated values that lie outside an integer pixel grid?
3. What coordinate systems should be recognized for dimensions that have a spatial interpretation, such as pairs intended to represent horizontal and vertical position within an image? Certainly Cartesian coordinates are ubiquitous, but one can also consider homogeneous (projective) geometry, polar representation, orthonormal vectors not parallel to image sides, and so forth. Even with the most common orthogonal axes (parallel to the image sides) there are multiple options for the origin point, including all four image corners and the image-center (with direction of increase typically upward and to the right). Should arbitrary Cartesian origin-points be allowed inside (or possibly outside) the image interior, other than the center? Where should the center be located for an image with even pixel-height and/or width?
4. Should it be possible to designate points via tuples representing coordinates under numerical transforms, such as logarithms, scaling factors, hyperbolic arcsines, or biexponentials?
5. Should annotations possess a zoom dimension that can vary independently of the ground-image zoom? That is, should annotations support a transformation where they are rescaled, while the ground image remains the same (e.g., a circular or polygonal shape expand or contract relative to its center), or should any such change be executed simply by updating the point- and/or length-set? Likewise, should there be a mechanism for representing the annotation-scale (whether or not it mirrors the ground image) at different moments in time, e.g., when the annotation was first created—as it is currently being viewed—or are these presentation data that need not be recorded in the annotation itself?
6. What scale units should be recognized for coordinate positions and intervals, such as the nine possibilities standardized in SVG (centimeters, inches, and so forth)? Should annotation-sets potentially mix two different such scales (e.g., centimeters and percentages against image width/height)?
7. How should annotations' shape point-sets be ordered? Should the ordering be left to the discretion of any human user or software component that defines an annotation? Should annotations that are identical modulo point-set permutations be deemed equivalent? How should point-sets with three or more colinear points be handled?
8. Should annotation shape be constructed solely via a point-set or should other magnitudes (what we last chapter called a "length-set") be allowed as a way of describing the geometrical qualities of the annotation shape?
9. How should special roles for particular points in a point-set or lengths in a length-set be described? Should roles refer to numeric positions in point- or length-sets modeled as ordered sequences, or should point/length sets be provided instead as key-value or key-multivalue associative arrays, or some combination?
10. Should annotations be able to refer to other annotations as a data point intrinsic to the annotation data, as would be the case with an arrow pointing to a separate annotation,

for example? If so, how should annotations be uniquely identified?

11. How should view-state data about annotations, such as zoom factors, colors, line widths, and cross-references between annotations, be registered as data structures complementing annotations as opposed to intrinsic data *within* annotations? Which properties are in fact intrinsic and which are presentational?
12. How should we represent annotation-sets or collections, and should such sets be nested? Assuming that many low-level details (involving scales, coordinates, point-set operations, and so forth) are defined by the environment in which annotations are constructed, and therefore applicable to annotation-sets as a whole (rather than being details of individual annotations), to what degree should individual annotations be capable of overwriting the default properties of the set to which they belong? Can a single annotation use a different scale of measurement, or coordinate system, for instance, or treat as intrinsic data visual details that would normally be considered presentational?
13. What mechanisms should be employed to describe calculations performed as part of annotation data and/or within parameters stipulated through annotation data (e.g., the length of a line segment)? How should we attach representations of arbitrarily complex mathematical expressions and/or algorithms that may be involved in such calculations?
14. How should different kinds of curves that could be algorithmically described and/or rendered be supported as annotation shapes? Should an annotation system represent only a fixed selection of curve-types or should the options be open-ended, with a mechanism for supplying external procedures to perform calculations on special curve-types whose mathematics are opaque to the system?
15. What varieties of calculations should be recognized by default as plausible operations that would typically be possible for annotations in general, such as area and perimeter, boundary-crossing counts, notions of geometric center (via minimum circumscribed circle, say), and so forth? Should the suite of such calculations be fixed *a priori* or could one facet of an annotation-environment or annotation-set be the collection of computations which could be activated as operations on typical annotations? In the case of user-designed curves, should implementations of calculations matching a particular purpose (e.g., area or perimeter) be required as a "contract" for recognizing a particular kind of curve as a valid shape-type?
16. How should textual data (whether free-form or constrained by controlled vocabularies) be represented when used to comment on or describe the role or the salient features of an annotation? Insofar as annotations are created to describe biological phenomenon for which the ground image is deemed to show evidence, how should annotation data be connected with data structures that describe such biological details in a rigorous fashion; e.g., through standardized terminologies or indicators, such as (say) **RadLex** diagnostic codes, or should connections between annotations and bioinformatic descriptions be asserted outside of annotations themselves?
17. How should details about the ground image be represented within the annotation, and how should the image that annotations target be identified? Should annotations targeting the same image be grouped together such that pertinent image-details (e.g., color depth and dimensions) are available as data within the group of annotations, or should that data always be provided through a separate object or structure representing the image itself?

18. Should annotations always refer to one single (ground) image or should there be a mechanism for defining annotations either as applicable to more than one image (in the context of image-processing pipelines, or perhaps multiple similar 2D slides from a 3D or 4D resource) or as targeting one image, which is a transformed version of some previous image (so as, typically, to facilitate the derivation of annotations or the extraction of features that the annotation calls attention to, for instance when analysis is performed on a morphological simplification of an original ground image)?
19. How should the provenance of annotations be described: factors such as whether they are created by people or algorithmically, what is their diagnostic (or explanatory or computational) purpose, their intended use (are they visual cues to observed features in an image or precise mathematical representations of image features or regions/segments)?
20. How should annotations model data which is best presented via derivative images with a similar format to the original image data, rather than via geometric structures such as shape data? For example, consider regions demarcated with a black-and-white or (for fuzzy regions) grayscale image overlaid on the ground image; thereby partitioning the ground into an interior point-space (black) and exterior (white). The purpose of a derived image in this case is to replace a geometric annotation-shape with a more granular and precise machinery to isolate distinguished regions. How can annotations defer their shape data to images in this sense?
21. How should annotations notate image features evinced in regions demarcated by the annotation, particularly in cases where the region is segmented specifically because it tracks the area where a given feature is present; e.g., region boundaries are the outermost points where a certain texture exists in (some neighborhood within) the image, or edges where the conformity of the image to a textural pattern crosses below some threshold? How should (potentially complex) computations or data structures endemic to image-features be oriented and connected to the annotation proper?

The above list amounts to circa 20 questions, or more given that some above items span multiple more specific questions. This is not necessarily an exhaustive list, but hopefully it indicates the scope of detail, which can arise when contemplating a general-purpose framework. It is also true that frameworks may adopt narrower scope, which could pre-emptively resolve some of the issues itemized above. For example, AIM does not deal extensively with image features or with computational geometry (apart from character-string notation of calculations), which obviates the need to bridge annotations with other kinds of computational data (at least within annotation-data proper). However, annotations are typically used in software environments that have a broader scope than annotations alone, so the questions we have identified may still be in effect, merely deferred to the software utilizing the annotation framework rather than encompassed in the circle of concerns articulated by the framework proper.

With that said, the problem of integrating with other data domains and broader software ecosystems is a good rationale for an annotation framework to address a relatively wider scope, which draws in the suite of questions we have identified. Assuming going forward that these are all consequential questions, then, we will make some general observations about them.

8.1.2 Procedural data modeling (and the limitations of ontologies)

Our first claim is that image-annotations provide a case study in the limitations of "Ontologies," at least in the Semantic Web OWL (web

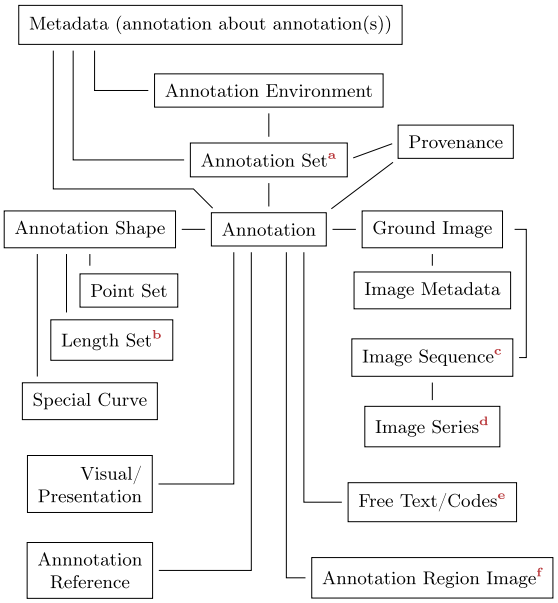


FIGURE 8.1 Diagram of annotation-associated datatypes. [a] Annotation Sets could potentially contain other annotation sets. [b] We use "length" set to denote the set of any magnitudes needed to geometrically specify the desired annotation-shape. [c] Image "Sequence" referring to transformed versions of a ground image (after grayscaling, morphological operators, or related modifications intended to make the image more amenable to analysis). [d] Image *Sequence* referring to logically related ground images. [e] Codes could be one or more diagnostic/prognostic terms selected from a Controlled Vocabulary or could be embedded as keywords in ordinary (free-form) text accompanying the annotation. [f] "Annotation Region Image" refers to a mask-image (e.g. a binary or grayscale image) intended to demarcated the contours of an annotation shape.

ontology language) sense (and certainly of specifications that might be designed to play a role akin to "light-weight" Ontologies, such as XML schemas via Document-Type Declarations).

To be sure, it is possible to define an Ontology-like network of classes associated with image-annotations (Fig. 8.1 outlines the principal classes that might be involved). But such an outline does not articulate the range of implementational possibilities that are intrinsic to the image-annotation domain (and summarized in part by the above 20 or so questions). Consider specifically, for example, the notion of annotation *sets* and annotation *environments*. The rationale for positing these classes is first that many low-level details concerning how annotations are defined (such as coordinate systems, the details of point-set construction, the range of recognized shapes, and so on) are unlikely to vary from one annotation to another. If annotations are constructed by a human user, for example, most of these details would probably be defined by the software that the user uses. If they are constructed via computer algorithms, the code libraries providing those algorithms might similarly define a basic quantitative model that would underlie the overall process; this model would presumably be shared among all annotations generated as such.

It would then be memory inefficient—and an inaccurate logical representation of the annotation framework—to present low-level details along these lines as data structures *within* individual annotations. Even if it were possible to override certain defaults on an annotation-by-annotation basis, the correct logical gloss is still that a specific quantitative model (reflecting issues like coordinate systems and shape-geometry options) is endemic to the environment where annotations are constructed and will engender details shared amongst annotations by default. These are, in short, properties of annotations grouped by a common origin or environment, rather than properties sited in single annotations (in the default cases).

This therefore points to one aspect wherein annotations can be logically grouped together: those that emerge from a common environment will tend to share low-level operational and quantitative details. However, there are other criteria through which annotations could be aggregates: a common ground image or image-series, or gathering all annotations on a group of images brought together by the human annotator. Presumably, one annotation *environment*

could include or engender many annotation *sets* in this sense. Insofar as every set would inherit low-level details from the environment where it is created, such details logically belong to the environment more so than any specific set. Moreover, annotation-sets can be collected for different reasons (e.g., a shared image or image-series) and could moreover be nested hierarchically; the criteria for aggregation and the parent/child relationships would seem to be points of information that are intrinsic to annotation-sets, but not to annotation-environments (insofar as the environment defines the basic operational and mathematical conditions wherein annotations are constructed). In this sense *sets* and *environments* play conceptually distinct roles, which is conveyed by modeling them as distinct classes.

However, such a rationale does not settle the question of what details should be modeled at the environment level or the set level, or indeed (to the degree that they may vary on a case-to-case basis) at the annotation level itself. Where, for instance, should the coordinate system (at least for default cases) and the Cartesian zero-point be defined—as a property of annotation environments or sets (or even individual annotations)? An Ontology could simply select a "distribution" of information which seems most appropriate for the widest range of cases. Ontologies need not be completely open-ended; it is the possibility of structural variations that allows there to be different Ontologies. But this example illustrates how structural choices are not isolated; the effects of choices specifically local to one "part" of an Ontology can propagate across the framework.

Assume, for example, that annotation *environments*, *sets*, and individual *instances* have a roughly hierarchical relationship: one environment produces many sets, and sets contain multiple instances. Low-level details would typically then be inherited from higher-up in the hierarchy downward. If information is distributed across all three levels, then it is only possible to use individual *instances* by taking information from the *set* to which they belong and the *environment* according to whose rules they are constructed.

In practice, this means that an application needs to obtain an environment object as a context for using an annotation *set*, and needs to obtain an annotation-set object as a context for using single annotations. Assume now that annotations are retrieved from a database: the query interface to that database would then have to be organized so that objects correspond to this order of initialization and inter-dependency. How should a query which intrinsically returns *one* annotation deal with the enclosing sets and environments? Should a query interface start by loading an environment, using that as a parameter to further queries, so that the handle to an environment object is prerequisite for a single annotation? Or should a query-result for individual annotations be augmented with indicators for the annotations' respective sets and environments so that those objects can be retrieved if not already? In the latter alternative, given query results including multiple annotations, how should data structures associating annotations with their sets/environments be described? Query types that are more complex (involving multi-layered structures, rather than individual values or simple list-like collections) require proportionately more coding steps to parse.

Moreover, should environment and set-level defaults be imposed uniformly, or should they be overridable down the hierarchy? The former option yields a framework that could be too rigid for some use-cases. On the other hand, suppose we opt to make a variety of implementation defaults overridable. This entails *first* that procedures must be available to construct the alternative setup where it departs from the defaults. And *second*, because it then cannot be assumed that annotations and sets thereof inherit low-level details ubiquitously from the environment, there must be a mechanism to flag whether a given annotation does or does not encompass

any such overrides, and, if so, to define them. This book's demo code allows annotations to include a "default override" object, which provides an interface for declaring an environment for a given annotation that differs from environment defaults in various ways. However, the demo annotation class also uses pointer-union types to restrict the memory-size of individual annotations, considering that many data structures needed by *some* annotations will be superfluous for many others. Analogous issues would come into play for databases persisting annotations: the flexibility of allowing numerous extra structures (for overrides, special curves, point/length-set roles, etc.) for *some* annotations has to be balanced by techniques for compactifying such structures when they are "empty" or unused vis-à-vis annotations where they are unneeded.

In short, choices such as how to distribute information across hierarchy levels tend to propagate to implementation choices involving memory organization, database queries, procedural requirements for data types that take on roles specified by Ontology classes, and software engineering in general. It is difficult at the Ontology level to represent the details of implementation choices, or the range of choices available to an application vis-à-vis specific concerns. One might reply that Ontologies are intended to be schematic models of domains, not rigorous blueprints for software implementations. That is true as far as it goes, but implementation choices determine the degree to which software components are interoperable: if a given Ontology can be realized via architecturally incompatible software, then Ontology alignment alone (as opposed to software-engineering and data-exchange protocols) is not sufficient for interoperability.

The idea of *protocols* serving as a contrast to *Ontologies* can be further illustrated with an example we alluded to earlier: consider the issue of annotations' point-set ordering. An annotation "domain model" can recognize different possibilities for enforcing or interpreting the order present among points in the annotation shape. These can take the form of axioms (all point-sets which are permutations of each other should be considered equivalent) or stipulations (point-sets should be automatically ordered during construction) or classifications (point-sets should be subtyped as clockwise, counter-clockwise, nonconvex, or self-intersecting, with different ordering-specifications depending on the subtype). However, implementation-wise these various options become concretized through groups of *procedures*: procedures governing how point-sets are modified upon inserting a new point, in cases where it is required that point-sets remain ordered; to compute the proper position for a point relative to an existing set; to permute a point-set into a proper order; to determine whether an ordered point-set is oriented clockwise or counter-clockwise; or to determine if a point-set is orderable in the first place in terms of vertices of a convex polygon. Similar comments apply to questions about colinearity: however that issue is addressed, any resolution depends on procedures such as identifying when a given point is colinearly between two outer points, and filtering intermediate colinear points out of a point-set when that would be appropriate.

These examples point to how image-annotations are a representative case study for the general phenomenon which we highlighted in Chapter 6: rigorous documentation of data models tends to depend on code models that instantiate them. In particular, code models can describe the roles and pre-/post-conditions on individual procedures as well as how procedures are interrelated into logical groups. Decisions concerning how information is distributed among annotation-environments, sets, and instances translates into the procedures used to construct an environment anterior to individual annotations being processed, and into procedures for overriding environment defaults when necessary. Decisions concerning point-set orders

and normalization become manifest in procedures through which point-sets are modified and geometrically examined. In these examples resolutions to concerns examined abstractly within generic data models can only be concretely documented at the procedural level.

This book's demo code represents one possible implementation of an annotation class; it is not a definitive example of how annotations *should* be defined, but it serves as a case study in the *kinds* of procedures that are intrinsic to defining a relatively general-purpose annotation class. This image-annotation class and its peers use source-code annotations to document logical relations between procedures, and also employ various binary-representation techniques to make annotation data memory-efficient. The resulting *code* model is more fine-grained than an annotation *data* model which is implicitly instantiated by this code; it would be possible to concretize similar data models with a code library that differs from our demo in many low-level details. As a result, it might seem that the particulars of our annotation-related classes are implementation details that should be separate from higher-level data models. The purpose of data models is less implementation-specific, but rather focused on defining common requirements so that components whose low-level implementations may differ can nonetheless interoperate and exchange data.

The more that data models are abstracted from implementation details, however, the more that particular implementations may need to provide bridge code, which marshals data into a common format for purpose of networking and data sharing. The degree of extra code demanded to get two distinct software components to interoperate tends to be proportionate to the degree to which their low-level implementations diverge from one another. The best data-integration protocols tend to be receptive *both* to abstract data models *and* to implementation concerns, in that sensitivity to propitious implementational design patterns can help protocols adopt formulations that minimize the bridge code needed for implementations to participate in the protocol.

8.1.3 Different aspects of image-annotation data

As we have proposed, in multi-aspect modular design, each module is responsible for managing software concerns that cut across multiple facets of software development. We are focusing on the four aspects of procedural exposure, data persistence, serialization, and GUI design (which together fit into the "Semiotic Saltire" schema that we introduced in Chapter 6). For the sake of discussion, we will assume that a module responsible for image-annotations adopts data models similar to what we outlined earlier in the chapter, with details involving ground images, visual presentations, and annotation sets/environments represented as contextual parameters coexisting with annotation data proper (see Fig. 8.1 for an outline).

Assuming that annotation data (and the relevant suite of contextual data) is organized along those lines, then, we can examine how this data model projects onto the different concerns of the "Saltire." Here is a summary:

Visual Objects and GUI Design First and foremost, of course, annotations have to be rendered against the background of their ground image. Analyzing GUI design patterns in a modular context requires some care, because it may not be obvious which GUI elements belong to which modules. In the simplest case, each module would be responsible for its own suite of self-contained GUI objects—particularly autonomous windows. That is, any given window within an application would be designed and populated with data entirely from a single module; insofar as the application integrates multiple modules, their coexistence would be manifest in (potentially) multiple windows being open at one time. In practice, however, modules may need to be more

tightly coupled than strict separation of windows allows, particularly in the GUI context. In the case of image-annotations, it would be reasonable to factor details of image *acquisition* outside the scope of the annotation module, so that the latter would not be responsible, for example, for finding a specific image within an image-series or archive, or giving users a chance to load a new image to replace the one they are currently given. Concerns for annotations proper would then be woven with handlers for user actions involving searching for and loading images themselves (to be discussed further later this section).

Serialization We discussed some issues related to serializing image-annotation data in the context of AIM (the Annotation and Image Markup project) in Chapter 6. As we then intimated in Chapter 7, image-annotation data models could potentially be broadened in scope and made significantly more flexible than AIM itself, which would require extending AIM's serialization model to incorporate a wider range of contextual data and annotation details. Fine-grained details of annotation serialization is outside the scope of this chapter, but we can make a couple of general points. First of all, we contend that the structural rigor of serialization formats should be enforced by client libraries as much as (or more than) via document-level constraints on serialized data. In the case of image-annotations, the primary role of serialization is to encode annotations within one application so that they may shared with other applications running elsewhere and/or later. Assuming that the receiving application also obtains a copy of the relevant ground image, the serialized encoding should permit the second application to reconstruct the annotations in exactly the same form as they were (within the original application) constructed and superimposed on the ground image.

The best way to achieve the proper alignment between sending and receiving components is to use the same code base on both ends; therefore to implement low-level serialization details, it would be good for image-annotation modules to provide a code library that can be re-used across different endpoints, insofar as annotations are shared in serialized form (in principle such a library should be isolated from and not dependent on the module as a whole). The canonical example of a serialization-deserialization cycle would then be a situation where two different applications both use the same code library as the specific locale for procedures directly responsible for constructing and/or parsing the serialized formats.

Of course, one rationale for curating a standardized serialization format is to free applications from depending on one specific code library. As a result, the module's serialization format should be designed to facilitate alternative libraries which could parse the same data. This is one reason why serialization formats often stress standard models, such as an XML Document Type Declaration (DTD), which can serve as a reference-point for developing multiple different code libraries that share a common serialization format. In general, confirming that a document (i.e., a file or a character or binary stream) conforms to a given serialization format is a separate process from actually parsing such documents, and can serve as a useful preliminary step (so that parsers can assume as a precondition that any strings they are presented with conform to the standard). For these reasons, formats (or meta-formats) such as XML are popular because these kinds of preliminary checks can be simplified by (say) DTD's; however, XML (and other popular formats, such as JavaScript Object Notation) can also be limiting in some respects. Modules could certainly turn to more flexible and expressive data formats instead. It is reasonable, however, in those cases, to provide code for document *validation* which is less complete than (and preliminary to) full

parsing/deserialization. In this manner alternative libraries designed for other programming environments (e.g., other programming languages) can mimic the validation code, separately from emulating (to whatever degree is appropriate) the actual parsers.

These comments apply to serialization aspects in general. For the more specific issue of image-annotations, note that data models in this context (at least insofar as they roughly follow our outline from earlier this chapter) are characterized by a relatively large number of default options that will generally be shared among multiple annotations, but with the possibility of many defaults being overridden on a case-by-case basis. In this scenario validation code for serialized documents may be relatively complex, because the core serialization for annotation proper may or may not have numerous residual structures encoding extra details. Techniques for managing variegated "shape constraints" in this specific context (see Section 6.4) are represented, albeit not with full rigor, in the demo code accompanying this book.

Data Persistence Questions about database representations for image-annotations can be intricate, because we have to identify which portions of annotation data are "queryable," in the sense we discussed in Chapter 6, that is, what parts of annotation data should be visible to queries against a database. Presumably, there would be few occasions where geometric minutiae (e.g., the precise details of annotation shapes) would be the subject of database queries. However, the text labels/descriptions and diagnostic information provided with annotations could certainly be represented in a queryable fashion. The simplest scenario would be encoding descriptions as ordinary textual data, but there is also a rather extensive literature on technologies such as "Semantic DICOM" [38], which associate annotations with diagnostic codes (including treatment plans) and/or controlled vocabularies (e.g., RadLex, the "radiology lexicon"). Such codifications are intended to make it possible to search large-scale databases of annotated biomedical images via diagnostic, treatment, or biomarker terminologies.

Annotation characteristics such as *calculations* lie somewhere on the spectrum of detail between text and/or controlled-vocabulary descriptions and granular geometric representations of annotation-shapes. Proposed Semantic DICOM protocols include the ability to query image data-sets for quantitative data that can be defined within (or via) annotations. A canonical example, used on the starting page of the **SeDI** project (https://semantic-dicom.com/starting-page/) is "Display all patients with a bronchial carcinoma bigger than 50 cm^3" (*see also*, e.g., [22], [37], [39], [11]). Calculations depend on fine-grained geometric details, so making this kind of quantitative data available for *queries*, where a query engine has to scan over many different annotations (without the option of reconstituting annotations individual to precisely examine their geometric properties) requires some functionality to classify calculations into queryable categories (e.g., "tumor size"). We will not examine this process in detail, but note that semantically this involves combining annotation-specific data with biological interpretations, and therefore belongs to the larger issue of representing biomedical findings warranted *through* annotations alongside annotations themselves. In this broader guise, we will return to this issue later in the chapter.

Procedural Exposure In keeping with our discussion in Chapter 6, a good starting-point for considering which procedures to "expose" for runtime reflection and remote-invocation scenarios is to look at those procedures that intrinsically implement, support, or operationalize important details of a modules' *data* model that informs its associated *code* model. At this point we can look at some concrete examples in the image-annotation domain.

As we reviewed above, one concern for a general-purpose annotation framework is how to specify the numeric and measurement dimensions applicable to annotations. There are several options, including using template classes which would be specialized on number and scale types (e.g., **double**s and millimeters, as one possible combination). Alternatively, classes within annotation data such as point-sets and length-sets could be modeled as base classes, for which dimensional details are unspecified, allowing subtypes to be implemented with specifics such as (say) **double**/millimeter. The demo presents a third option, namely encoding a variety of dimension/numeric configurations within individual data types, which we will reference momentarily for sake of discussion.

The basic idea here is to use flags and/or enumerations to differentiate number/scale possibilities when these are ambiguous. For example, the demo code uses quasi-real types (which can be **typedef**'d to something other than **double**) for length and coordinate values, but supports a flag restricting these values to integers. When (but only when) that flag is in effect, any procedure initializing a length or coordinate value is truncating to an integer.[1] This is an example of procedural locations enforcing a data-modeling choice, and the procedures that carry out these checks (as well as those manipulating the relevant integer-only flag) are likely candidates then for exposure. Similar comments apply to measurement-units. The demo assumes that lengths and coordinate values can be provided in numerous different units, which are an intrinsic part of the value,[2] but also provides procedures to convert each value to a different choice of units, and to synchronize two different values when it would be unreasonable to have a mixture of disparate units (e.g., for arithmetic operations and for composing coordinate pairs; one operand or element of the pair is recalculated in terms of the other's unit scale).

With respect to GUIs, determining the proper scope of an annotation module's responsibilities is complicated by the fact that image *annotations* might be separated from management of *images* themselves. Annotations and ground images obviously have to be displayed together, but other software-engineering aspects between the two facets (images versus annotations) do not necessarily coincide; serialization and data persistence for annotations can be separated from the corresponding operations for ground-images, apart from the annotation maintaining a reference to the corresponding ground image through some sort of identifying code or file/web location.

This leaves open the question of how exactly to implement GUI components when the visual artifacts seen together may be the responsibility of distinct modules. It would be theoretically possible for multiple modules to interact within single GUI components; for example, consider a scenario where the ground image is a node on a **QGraphicsScene** object (this assumes that the relevant applications is composed in C++ with the Qt GUI libraries). Annotations can then be further nodes drawn on the same graphics scene, and procedures in the annotation model can be presented with (a pointer or reference to) the **QGraphicsScene** instance, without managing other GUI objects.

For the sake of discussion, however, assume that the image-annotation and ground-image handling modules are more strictly separated, and that the annotation module handles its own GUI windows. In that sort of setup, this module would receive an object encapsulating a ground image (or perhaps a binary package with the raw image data), but would otherwise work in isolation to display the image, and then render annotations with reference to it. Most image-management operations, on the other hand,

[1] By "coordinate value" here we mean one value in a coordinate vector, canonically x and y as pair-elements in an image location.

[2] That is, each value encodes a magnitude and also a code marking which measurement scale to apply for it.

would be delegated to separate modules responsible for images themselves, as will be outlined in the next few paragraphs.

8.2 Annotations and radiomics

We feel that image-annotations serve as a good case study for issues surrounding modular design, which is why we have devoted space to annotations and their associated data structures. However, the practical consequences of modular design-choices may be more apparent when considering the larger bioinformatic systems wherein annotations are a relatively small part, so we turn to this larger picture next.

8.2.1 GUI operations involving images and image-annotations

An annotation module meeting the general profile just outlined—maintaining a self-contained window showing ground images and annotations drawn on them, but delegating image-management to other modules—would presumably need to be paired with modules through which users find images of interest in the first place. Applications would invoke procedures in the annotation module only after loading images requested by the user through a database query, file system search, or simply opening a local image file.

In this scenario the annotation module would have capabilities to display each ground image itself, but in general would not implement separate GUI classes for other operations related to images proper (as compared with operations pertaining to actual annotations, that are its proper domain). For example, the annotation module might skip implementation of functionality to open a new image (replacing the current ground image with a different one), or to search for images (in a database or file system) that users would load to replace the ground image currently viewed.

With that said, however, it would certainly be plausible that responsive GUIs would allow users to indicate the desire to load and/or search for a new image via actions performed on the GUI objects managed by the annotation module. For example, a context menu activated on points within the background image (at least those not covered by an annotation) could have, as one of its menu items, the option to look for a new image to load in lieu of the one currently being rendered (along with its annotations). The module would in that case, presumably, indicate via a signal-emission or procedure-call that the user intends to search for a new image, and delegate to other modules or components the process of showing dialog boxes or similar tools responding to that request.

Assuming this overall plan for routing requests, there are then two different scenarios where an annotation module would be rendering a new image and its annotations (new in the sense that the image is not currently visible on-screen). One scenario is that the module was not active previously (during the current run of the surrounding application, anyway) or at least that any windows showing ground-images have been closed. An alternative scenario is that the module is managing an open window with its ground image and annotations initialized, and receives a signal that the user now wishes to view a different image which the user has opened or selected (potentially that signal could be indirectly a response to an earlier signal from the module itself asserting the user's preliminary desire to switch images). Identifying these two different sorts of "entry" scenarios can be useful for building a procedural model relating to the module's "procedural exposure" concerns.

Fig. 8.2 sketches out a few of the more important procedural steps that would be involved in proceeding from when the module receives an identifier or data structure for a new image to the point where this ground image (and its annotations) is rendered. This is not a very

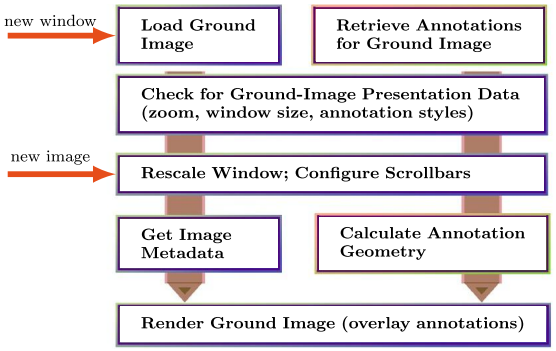


FIGURE 8.2 Entry points for hypothetical image-annotation module.

schematic picture (leaving out many potential steps and not really clarifying the branching logic between showing an initial image and replacing the image on an already-visible window), but it is intended simply to give an impression of how procedures can fit together into logical sequences, and also how a module may have specific "entry" points. By this terminology, we mean that at entry points the module is receiving data from an external source and is also receiving notice that some procedural chain should be initiated. In this running example, there are two kinds of "entries" in this sense: one is initializing a new window with the image-data provided, and the other is replacing the image with another viewed within the former image's windowing context.

In general, it is worthwhile to tag every point where a module may receive data from an external source; it is also worthwhile to tag where an external source may trigger (or request) a chain of actions that the module implements. An "entry point" would then be a location (e.g., a join-point; see Chapter 6) where both of these conditions are met. Rigorous procedural models should accordingly identify such entry points, and the code should be designed to facilitate this process. One way to do so is providing specific procedures for each entry point (i.e., each site in a logical model of what the module should accomplish that would serve as an entry point). The entry-point procedures might call other procedures, of course, but providing entry-point procedures that encapsulate the initial steps in a "logical" picture of modules' functionality (abstracting from implementation details) can then make it easier to trace the "flow of information" within the module. Moreover, once entry-point procedures are specified and identified as such, they are obvious candidates for being exposed to scripting, testing, and remote-invocation engines.

Many systems exist to diagram procedures in a module (or code library, software component, or analogous body of interrelated code) and the logic of how data passes between them, as well as how groups of procedures follow in sequence to perform a specific task.[3] Analyzing these sorts of diagrams (their vocabularies and the attributes they may identify per procedure) is outside the scope of this chapter, but suffice it to say that once a procedural model is created (perhaps informally), such a model can serve as a template for establishing a module's "procedural exposure." Procedures which appear to be particularly important in recurring sequence-patterns are likely to be those which could be targets of script-based adapters, unit tests, and other scenarios where such procedures might be invoked dynamically. Likewise, the collection of exposed procedures serves as an introduction to the overall functionality of a module, so that procedures that seem most functionally important in a module should be exposed partly because they provide an overview for developers trying to become familiar with the module's implementation.

Image annotations have only limited use in isolation; in contexts related to bioimaging, the main purpose of annotations is to call attention to biomedically significant image features,

[3] As examples—without implying that these formal systems align completely with our summaries here—consider Petri Net techniques mentioned in Chapter 6, or, say, [15], [27], [5], [35], [23], [13], [36], [17], etc.

which are identified either by a human expert evaluating the image or by image-processing algorithms. Once identified, these image-features suggest additional facts about the biological material sampled within the image, and accessing such non-image data requires capabilities that presumably lie outside the scope of the annotation module proper. Accordingly, this module would need to provide users opportunities to follow up on image-view focused actions via those handled by other modules.

Because image markers can be one of the more effective means of discovering and presenting biomedical facts or predictions, hypothetical application-sessions where users start with capabilities provided through an image-annotation module is a reasonable case study for analyzing issues in how modules would interoperate. This provides a rationale for emphasizing image-annotation as we have here, though the remainder of this chapter will transition to other forms of biomarkers.

8.2.2 Image processing in the context of broader-scale workflows

Bioimage analyses are usually most valuable when they can yield relatively simple data structures that have unambiguous biomedical interpretations. For example, image segmentation in the context of cervical cancer yields a classification of imaged nuclei into different bins, where one category corresponding to likely cancer cells, typically on the basis of one or two derived parameters, such as measurements of nuclear enlargement and deformity (relative to healthy cells).[4] Similarly, the density of blood vessels within tumor microenvironments can be estimated via "fractal dimensions," which measure the cumulative length of blood-vessel structures connecting tumors to surrounding tissues.[5] In general, we want image biomarkers to yield statistical measures from images which correspond to bioinformatic quantities yielding diagnostic and/or prognostic results: What proportion of cells in a blood sample are cancerous? How aggressively has a tumor started to "colonize" surrounding tissue? Where has scarring diminished blood supply to damaged heart tissue? How extensively has SARS-CoV-2 infection compromised lung functioning (visible due to "ground-glass opacity")?[6]

Often these sorts of biomedical questions, for which bioimaging can provide partial answers, may also be answered in part by other means, such as genomic tests or biochemical assays. This opens the possibility of bioimaging and (say) biomolecular results mutually reinforcing one another. For example, immunohistochemistry staining (IHS) is often used in biopsies diagnosing many forms of cancer: antibodies are introduced into a tissue sample, causing the sample to be stained whenever specific proteins are present that bind to the introduced antibodies. There are hundreds of different IHS antibodies that can potentially be utilized, depending on which specific proteins are targeted. Assessments of the quantity of target protein evident in a sample can itself depend on image-analysis, usually comparatively simple measurements of the degree and intensity of sample staining. Immunohistochemistry (IHC) and IHS are one of several antibody protocols; flow cytometry, for example, provides an alternative (*see*, say, [40] or [24]). In this context immunostaining is based on immunofluorescence antibodies, rather than visible staining, resulting in cells containing the target proteins emitting fluorescent signals that are detected using FCM equipment.

The visible final stages of an immunostaining assay may therefore be a microscopy image showing obvious staining patterns that can be

[4] See Section 4.2.2 earlier in the book for references.

[5] *See*, e.g., [42], [9], [14], [43], as well as [1, page 4] and [34, page 6] cited last chapter.

[6] *See* [20], [29], [31], etc.

graded by human or software analysts, or may potentially be FCM plots, where each cell corresponds to an "event" wherein the cell has generated fluorescent signals captured by an FCM channel (this data can then be modified via FCM methods, such as dimensional transforms and gating to yield an intuitive visualization of the event-data).

The resulting visualizations, however, are only the end stages of complex assays that require disciplined lab methods during sample preparation and analysis. Bioimaging in the context of biochemical assays is different from non-invasive image-processing using radiology, for example, to picture solid tumors. With diagnostic assays, modifications are induced within a tissue sample to control how the sample's physical properties (e.g., the presence of one or more specific proteins) becomes visually expressed under bioimaging. Scientists can physically manipulate the samples to be imaged, instead of or in addition to tweaking the image-analysis process, to improve diagnostic accuracy. At the same time, this means that data concerning how tissues are sampled and prepared needs to be modeled alongside of the image-data proper.

In a paper devoted to more precise IHC quantitative methodology, for example, [18] (also discussed in Chapter 4) presented the following argument:

> The commercially sourced antibodies in [IHC] protocols often vary in their specificity and sensitivity, thus requiring meticulous optimization and testing. Hence, there is a need to enable users to rapidly screen parameter spaces to determine practical assay conditions for the specific biochemicals used. The ability to *quantitatively characterize* detected signals, perform *multiplexed detection of various antigens simultaneously*, and *rapidly implement IHC protocols* that are standardized or modified according to user need are facets of research that would be essential to fostering the next generation of methods in cancer research and diagnostics. ***(page 1)***

In other words, the authors are implicitly calling for greater standardization *both* of "assay conditions" and assays' "parameter spaces" *and* of the algorithms generating their quantitative results. For their specific workflow, [18, page 3] identify several adjustable parameters within the assay protocol, such as temperature, "the flow velocity of the processing liquid" and "incubation time" (because these authors' methods depend on a two-step binding process involving two different antibodies, the ratio of the second and first incubation times is also a consequential data point (page 8)). In a prior article [12], the same authors also discuss novel quantitative techniques for measuring assay results, such as what they call "saturation approach matrix" (SAM). The SAM metric considers not only staining intensity once antibodies have been maximally bound to their target "analytes" (e.g., proteins), but also the rate at which these reactions occur. In this sense, SAM data is not only a form of image-biomarker derived from static images, but an (indirect) measurement of biochemical reactions occurring prior to the assay's final state (whereas conventional analyses would consider only the final state in isolation).

The quantitative techniques introduced in [12] depend on "microfluidic probes" (MFPs), which can freely move over the surface of a sample and (if desired) modify the sample at specific points, e.g., by introducing new material, while also picturing a local region of the sample around its current target point via an inverted microscope [10], [32], [3], [7], etc. Although MFP devices capture image-data via integrated microscopes, they may also be equipped with sensors that read signals in other media; [28, page 7], for example, propose "MFP ... combined with microelectrode array technology to study the electrical changes in neuronal networks in response to topical application of neuromodulators" (they cite in turn [26]). The MFP probes, in turn, are one example of "biosensors" that collect data about tissue samples by detecting optical, electric, or fluorescent signals. Whereas older biosensors tended to be fixed in place, recent technologies such as MFP enable

probes to freely move around a sample's surface. In some cases, biosensor data is mapped against single locations in a sample, so that the data has characteristics of a bioimage (wherein individual pixels roughly correspond to individual "points" on the sample). However, this image-like data is not necessarily acquired via the same physical mechanisms as a microscopic or radiographic picture. For example, a popular class of sensors is constructed according to the physics of "surface plasmon resonance" (SPR), where sensors detect "modulations" in patterns of light waves' resonance against a biological sample (when it is introduced into a specific configuration of instrument layers involving glass and gold) [41]. The "picture" that emerges infers properties of the underlying sample via properties of light waves, but instead of only capturing wavelength (color) as in an optical microscope, SPR results in more complex refraction-modulation data, which are then subject to digital processing.

As examples such as SPR and fluorescent flow cytometry illustrate, there are a wide range of physical mechanisms by which properties of a biological sample can be investigated via properties of light waves that emerge from or interact with biological materials. What we conventionally call "images" are only one manifestation of the larger principle that information about objects is encoded in the light that they emit, refract, and/or reflect. Consequently, it becomes unexpectedly difficult to identify the proper scope of domains, such as "bioimaging" and "image annotation."

Should *images* in these contexts refer to the classical sense of a "picture," which encodes information via *colors*, with the color of a single pixel being considered (as an idealization) the color of light reflected from a miniscule patch on the surface of the imaged medium, analogous to human vision? Or should we accept a broader notion of *images* encompassing a wide range of technologies that acquire spatially extended data about a sample by probing light waves for different physical properties, not only in the sample's "natural" state, but potentially after probing the sample with external energy-sources, such as lasers? Moreover, what about physical phenomena *other than* light waves, such as piezoelectric sensors?

These questions are not philosophical speculations on the "nature" of an image; more concretely, they address problems of integrating biomedical information acquired from a heterogeneous suite of devices. If we remain within the context of "classical" images, e.g., those derived from optical microscopy or radiography, there is a relatively well-defined ecosystem of digital formats and software capabilities, reflecting shared assumptions about the nature of "image data" which different software components will all be acting upon. For example, images are encoded within several canonical formats, such as JPG, PNG, and TIFF, and there are specific kinds of meta-data which are ubiquitous as preconditions for properly reading image data, such as dimensions/resolution, color depth, and the orientation of orthogonal axes relative to the pixel matrix. Analytic notions, such as regional contours, morphology operators, color transforms (such as grayscaling), image-masks, color averages and interpolations, region areas and perimeters, and so forth, are relatively consistent across different image formats and image-processing methods. We can, for example, ask about (say) the area of a region measured as a proportion of the area of its minimum enclosing circle, or about the morphology of its convex hull, whether we are calculating these results via OpenCV or ITK (to mention two popular image-processing libraries) and whether the underlying image is saved as a PNG or TIFF file.

Because of the relative consistency among different image-processing and image-viewing applications—if we stay within the "classical" imaging scope—it is possible to investigate image-analysis pipelines as generic workflows abstracted from the specific software tools with which the workflows would ultimately be im-

plemented. Moreover, analytic libraries can be paired with image-viewing software and image-acquisition sources in different combinations. The process of obtaining raw image from, for example, a DICOM image series, subjecting those images to a predefined analytic method, and visualizing the results via image-viewing software, is sufficiently standardized that some variant of this methodological outline can be implemented on a wide range of components. This standardization allows users' understanding of image-related software to largely carry over to other software (even if it is implemented in a different programming language, distributed in a different commercial or open-source environment, and so forth), and allows image-related software components serving different roles to interoperate.

Widening the scope of "bioimaging" *outside* classical image-acquisition modalities, on the other hand, has the effect that this degree of standardization and interoperability becomes diminished. The terminology and mathematical models for manipulating "image" data deviates from "classical" images the more the physical mechanisms of acquiring this data deviate from microscopy or radiography. For example, flow cytometry gating evinces idiosyncratic terminology and quantitative frameworks despite the fact that geometrical principles behind gating overlap in many respects with concepts from image annotation (leading to proposals in the FCM community, for example, to integrate their own data models within DICOM; [16, page 1, e.g.] argues "The large overlap between imaging and flow cytometry provides strong evidence that both modalities should be covered by the same standard"). As we suggested earlier in the chapter, expanding the scope of data models too far runs the risk of such models being over-complicated with special use-cases and special-purpose extensions that are tangential to the core foci of the model. On the other hand, failure to synthesize interrelated data models can result in the unnecessary "fragmentation" of software ecosystems that give rise to (and are shaped by) data models, as we argued in Chapter 4.

8.2.3 Data profiles for annotation and image markup

Chapter 6 sketched the rationale for **aim-client**, a code library included as supplemental materials for this book which facilitates the use of data sets employing AIM annotations. This section will continue that discussion by delving further into the AIM data model and how we try to refine certain elements of AIM's semantics with **aim-client**. In practical terms, **aim-client** is first and foremost a utility library for deserializing AIM data. We demonstrate **aim-client** through sample data sets published alongside this book, where **aim-client** can be used to programmatically study **AIMLib**, aside from the extra features added by **aim-client**.

Since it was standardized, **AIMLib**'s adoption in radiology and related bioimaging fields has been driven mostly by clinical software that adopted the AIM format (as one mode for exchanging annotation data, among others, such as DICOM-SR (DICOM structured reporting)). We are not aware of software packages that provide access to AIM data in a standalone fashion, outside the context of a larger bioimaging application, which could potentially stymie researchers who wish to examine library code directly to get a more thorough grasp on AIM data and architecture. Potentially **aim-client** can help in this regard, because **aim-client** sets up an environment where AIM-compliant data (such as XML files) can be read and parsed, yielding instances of AIM classes that can be examined at runtime (e.g., through a debugger).

This book's republished data set is bundled with code in the form of a Qt project which links against both **AIMLib** and **aim-client**; users can accordingly load sample AIM-compatible XML files in a Qt environment. For example, a natural way to use the **aim-client** project is by running the

code in Qt creator, a C++ Integrated Development Environment (IDE) particularly targeted at Qt projects. The IDE's debugging features (for setting breakpoints, examining local variables, searching the code base for symbol-names, and so forth) can then be exploited to decipher AIM data structures after they have been initialized from XML files.

Given a properly formatted XML serialization, then, we can use the AIM **XmlModel** class, which provides a **ReadAnnotationCollectionFromFile** method that returns an **AnnotationCollection**. This latter type is a base class whose subtypes differentiate annotations-on-annotations (metadata) from annotations-on-images, which are our primary concern. The **ImageAnnotationCollection** type is a convenient way to accommodate the fact that some images may have multiple image-annotations (multiple regions-of-interest), even though in practice an image may have only one, so the "collection" is actually a holder for one *single* annotation-instance (this is the case for all the files in the sample data set). Most of the important structure, then, lies with this **ImageAnnotation** class. Each image annotation is a holder for collections of several object-types including "image references," which connect the annotation to the image it annotates; "annotation statements" that assert the annotation's significance (i.e., its biological/diagnostic rationale); and "segmentation" or "markup entities" (which carry information about image-regions defined as geometric objects on or segments of the image).

In addition to the actual annotation, AIM needs to account for many data points concerning the utilization of annotations in a diagnostic setting, such as diagnostic codes, confidence-levels that the image-interpretation is correct, metadata concerning the graphics properties of the image itself and how it was acquired, biological descriptions of the phenomena or entities (such as cells, tissues, lesions, etc.) visible at the region-of-interest (assuming correct interpretation), and so forth. As a result, the overall AIM data model encompasses many data types that are not explicit representations of annotation geometry itself. However, to organize the overall data model exemplified via AIM, it is useful to start with the model specifically representing geometrically described annotations, and then introduce diagnostic and biomedical metadata as refinements of that core model. In **AIMLib**, these fundamental data types are implemented via subclasses of a base **MarkupEntity** class. These subclasses, such as **GeometricShapeEntity**, are therefore a natural starting-point for examining the profiles of AIM data.

For the sake of discussion, we will mostly consider images and corresponding annotations in 2D (the 3D cases are similar, but harder to visualize). Positions in **2**-space are represented by AIM's **TwoDimensionSpatialCoordinate** class. Unlike other libraries that work with spatial data, such as the Computational Geometry Algorithms Library (CGAL), AIM only recognizes a single scalar magnitude for spatial intervals (a **double**, i.e., double-precision floating-point number). This appears to be an implementational choice, not a "semantic" one, in the sense that encodings of spatial regions and magnitudes can potentially utilize a diversity of quantifying strategies and coordinate systems. For instance, CGAL supports both normal Cartesian coordinates and also "homogeneous" coordinates—which are based on projective geometry—and allows coordinate systems to be parametrized on different kinds of scalar values, such as integers rather than pseudo-reals (which arguably better matches the image domain for many purposes, since one cannot have a fraction of a pixel); or rational/floating-point numbers with varying degrees of precision (a simple example would be using single-precision **float**s in lieu of double-precision, which makes geometric representations more memory-efficient, or in the other direction using more exotic numeric types for greater mathematical precision) [6, page 14]. Also, some image-analysis algorithms are ex-

pedited using polar coordinates [25, for example], implying that polar-valued annotations also have a role in documenting image-features tagged by those algorithms.

These comments are tangential to AIM *per se*, because a **double**/Cartesian coordinate framework is probably the most sensible default option, and AIM use-cases may not warrant generalizing the code for other coordinate options that would rarely be leveraged. Nevertheless, this example points to how even a seemingly simple construction such as two-dimensional spatial points can encompass a fairly detailed space of semantic alternatives. Code libraries that seek to operationalize a relatively thorough semantic model of the space-coordinate domain, encapsulating the diversity of representations that are computationally useful in different contexts for designating spatial points, regions, and magnitudes, would need to recognize a wider range of coordinate systems and numeric types than exemplified by AIM.

This case illustrates the kinds of situations where data-mismatch problems can arise: exporting data back-and-forth between **AIMLib** and other software components might require bridge code to handle coordinate conversions. Data-integration projects for which AIM annotations represent one data source would correspondingly need to anticipate the possibility of mismatches involving coordinate systems and the need for suitable bridge code as part of the integration workflow. Since AIM does not internally support a choice of coordinate-systems, using **double**/Cartesian exclusively, AIM also does not explicitly notate its choice of coordinate systems, so that it takes some exploration of the **AIMLib** code to grasp what may be necessary for interoperating **AIMLib** with other libraries.

In any case, 2D coordinate vectors then define geometric shapes straightforwardly. One noteworthy design choice is that the class representing 2D points (likewise for 3D) includes an extra data field asserting the "index" of that specific point in the coordinate vector to which it belongs. Depending on the geometric shape spanned by the vector, coordinate indices could have a fixed patten; for example, AIM describes circles by asserting the location of the center point, and then a location on the circumference. By giving coordinate points a predetermined order, shapes such as circles and ellipses can be defined by a simple coordinate vector, rather than by assigning separate data fields for points playing specific roles, such as centers or focal points; the coordinate index for each point implicitly indicates its role, and fixing this index for distinct roles ensures that the coordinate vector unambiguously encodes the respective roles (a circle center cannot be confused with a circumference-point, for example). The AIM shape subclasses provide their own **enum** types mapping indices to roles, so that code using these types can refer to the indices via descriptive names (e.g., **CenterPoint**), rather than an index number.[7]

With this system (perhaps designed in part to facilitate interoperation with DICOM Structured Reporting), AIM recognizes five principle types of shapes, mimicking DICOM-SR [4, page 94]: points, circles, ellipses, multipoints, and polylines. The difference between the latter two is that polylines must be closed (there is a presumed line connecting the last point in the collection to the first). These shape-types are derivatives of AIM's **TwoDimensionGeometricShapeEntity** class (the 3D case is similar, but supports additional "polygon" and "ellipsoid" types, the former enforcing that vertices be coplanar) and are also identified via an **enum** naming each of the five options, so given a pointer to the **TwoDimensionGeometricShapeEntity** base one can determine the nature of the shape represented by that object (however, given a generic

[7] Shape-designations that rely only on point-declarations to construct the relevant geometry do not need an extra notion of "length sets" along the lines we discussed in the last chapter; and indexing point-sets by roles alleviates the need for a separate role-key-value mapping, although these choices arguably limit annotations' flexibility to some extent.

MarkupEntity, which has several different subclasses, one cannot determine without type-reflection whether the encoded shape is 2D or 3D, for example). The range of shape-types is therefore expressed both by the **ShapeType** enumeration values and by subclassing the generic (two or three dimensional) base. Note that hypothetical extensions to AIM intending to introduce different shape-types would need to modify these **enum** values as well as provide the appropriate subclasses.

As mentioned in Chapter 6, AIM's coordinate vectors for representing geometric shapes is logically separate from graphical declarations affecting how the shapes are visually displayed. Properties such as line width and line opacity, accordingly, are treated as visual artifacts that are not intrinsic to the corresponding annotation—there is no notion of lines, or in general one-dimensional (potentially curved) paths, with an optional measure of line-thickness, being a form of annotation in themselves. The **GeometricShapeEntity** class provides a range of information *other than* the actual coordinates; in this sense **GeometricShapeEntity** is more general than **TwoDimensionalGeometricShapeEntity** (or its 3D equivalent), but is not a base class of these, being rather an ordinary data member sibling to the coordinate collection.

In addition to visual/presentation details, **GeometricShapeEntity** supplies data (and meta-data) concerning *calculations*, such as lengths, areas, and volumes (more complex calculations are also possible, such as the area difference between an enclosing and enclosed circle [21, page 698]). In addition to notating the calculation results, AIM supports meta-data explaining the purpose (e.g., diagnostic significance) of the calculation, including "QuestionTypeCodes" based on one data point stipulated in the **iso 21090** health data exchange standard.

As this overview demonstrates, geometric, visual/presentation, and biomedical data tend to be woven together in the context of bioimage annotations. For example, consider a simple one-line annotation with a given length and start/end points. The vertices themselves are *geometric* data, whereas the bioinformatic interpretation accorded to the length-calculation depends on the image's biological context (e.g., that the length of the line measures the width of a tumor or lesion, say). Extrinsic to both geometric and bioinformatic details, the "presentation state," or visual display parameters, governing how an annotation is currently being viewed (or should be viewed by default) within an image-viewer, represent graphics data (colors, line-widths and opacity, and so forth) which should be consumed by image software but is not significant for interpreting the annotation itself. This visual data would determine how the line is rendered when viewed as a graphic superimposed on the underlying image.

Continuing with data vis-à-vis one line segment, the *scale* through which the line's length would be interpreted depends on the image-resolution and on the image-acquisition process. For example, a one-centimeter length would correspond to a line segment 1 cm long when viewed on the image at its regular size (the visible segment would be a different length if the image is zoomed in or out). In the event that an annotation is performed by a radiologist viewing the image at a different scale, the visual form as seen by the annotation's creator would need to be scaled accordingly (AIM does not appear to support a record of the viewing conditions under which annotations are first made, information that could potentially be relevant for double-checking the original work). Moreover, 1 cm *on the image* would have different interpretations as a length within the actual tissues (or organs or microscopy slides) viewed through the image. All of these details surround a single-line annotation, which is geometrically the simplest shape (other than a single point); similar comments would apply to more complex annotations as well.

As this review illustrates, a data model for image annotations will actually encompass several

different parts, which are mostly autonomous from one another. One could picture an annotation data model as lying at the *intersection* of several larger data models, each of which have semantics that overlap but also extend beyond the concerns of image-annotation itself. These various domains—including details of images' optical properties, their biomedical/laboratory provenance, their clinical origins, and so forth—would be represented in finer detail than provided by AIM (or similar annotation frameworks) in the context of their own domain-specific applications; image-processing software for image biomarkers, for example, or decision-support systems for clinical data. Such applications would employ AIM as a data-sharing mechanism when needing to export or import image annotations in particular; but because each application has its own domain focus and internal data models targeted at their specific domain, it is likely that applications' information would need to be restructured to some degree to match AIM's serialization format.

Insofar as there are at least four larger domains which "intersect" vis-à-vis image-annotation—shape geometry, image-acquisition and image-encoding details, visual/presentation environments, and clinical context—there are at least four areas where "bridge" code may be needed to marshal data between **AIMLib** and applications using AIM as an image-annotation standard. Each situation along these lines, for which implementing bridge-code is unavoidable, presents a context where data integration may require special-purpose programming, rather than following seamlessly from the coordination of inter-related data models. Because it was formulated to serve as a common standard for biomedical image annotation in general, AIM facilitates data integration in that specific context. However, its influence as a data-integration paradigm is bounded by the scope of its primary focus (on bioimage annotation in particular).

8.2.4 Tradeoffs between data models' narrower and wider scope

Our point here is not that AIM is too narrowly focused, but rather that it provides a case study in the trade-offs between standards' complexity and their scope as data-integration tools. In particular, AIM coexists with other formats that address concerns related but not identical to image-annotation. For example, *gating* in the context of flow cytometry uses geometric regions as classifiers for cytometric plots (these were discussed in Chapter 4), which are geometrically similar to image annotations, but the underlying data takes the form of cytometry event matrices rather than image pixels. Accordingly, flow cytometry has its own coordinate systems and coordinate transforms, with domain-specific encodings, such as GatingML. Similarly, image analysis using computer vision can result in feature-vectors, which are analogous to image-annotations in that they are superimposed on underlying pixel-data and often represent segments or regions-of-interest in an image. However, image-features can represent patterns that have mathematical qualities distinct from the geometric shapes, calculations, and textual descriptions characteristic of image-annotations. Image feature-vectors and cytometric gating are examples of domains that are similar to image-annotation proper, but sufficiently removed from annotation concerns so as to require their own data structures and processing algorithms.

In the last two chapters, we have examined more widely scoped image-annotation possibilities that could potentially integrate multiple data profiles that are in some sense "similar" to AIM-style annotations proper, such as image features (and image biomarkers interpreted from them) and cytometry gating. Wider scope makes data-integration paradigms more substantial, because there is a commensurately wider range of scenarios where data conformant to the relevant standards can interoperate, but

wider scope also makes standards more complex and harder to implement.

The case of AIM illustrates how data standards' scope is typically driven by specific use-cases; for example, AIM was motivated by problems in sharing annotations derived from diagnostic environments, particularly those created by pathologists and radiologists to explain diagnostic findings. Neither cytometry statistics nor AI-driven image analysis are directly relevant to this specific genre of diagnostic workflow. It is therefore understandable that AIM's data model would not incorporate the kind of details that would be necessary to include (say) cytometric and image-biomarker annotations within its overall scope.

Nevertheless, it is certainly possible that *applications* using AIM would also need to recognize cytometric and/or image-biomarker data. For example, consider software to manage patients' clinical records, which attempts to provide visual tools for multiple kinds of diagnostic evaluations that were used for a given patient. Such an application might certainly benefit from distinct components to view flow cytometry plots, image-annotations, and image-analysis feature summaries, insofar as all three of these analytic modalities supply various kinds of clinically relevant biomarkers. Similarly, an application for viewing biomedical research data sets might want to encapsulate capabilities for importing and displaying data sets derived from cytometry, automated image-processing, and manual image-annotation respectively.

General-purpose software as just described would therefore be working with three (or more) analytic and imaging domains, which are similar in some ways but noticeably different in others. Such a mixture of similarity and divergence raises its own architectural questions. Certainly applications could support distinct data domains via separate modules: a clinical or dataset visualization program could incorporate AIM for annotations, ITK for image-processing, and libraries such as **cytoLib** or **immunoClust** (both part of **bioconductor** [8], [2], [33], [19], [30]) for cytometry. They would therefore have capabilities to read and manipulate data structures in each of these domains, but only in isolation from one another.

The problem here is that the domains *do* overlap in some nontrivial respects. For example, the GUI logic wherein a user can modify an annotation (by dragging GUI handles targeting annotation elements, such as points and lines) would be very similar for AIM annotations and for cytometric gates. Likewise, the GUI logistics for displaying image metadata could well be identical for the cases of image-annotation and image-processing/radiomics. Keeping the annotation, processing, and cytometric components fully isolated would therefore result in significant code-duplication in contexts such as GUI (similar comments could be made for database persistence).

Recall our picture of the "Semiotic Saltire" from Chapter 6: data structures tend to expand outward to encompass concerns such as database integration and GUI design. Continuing the example of image-annotation, radiomics, and flow cytometry as three related but distinct domains, the structural differences among their *core* data models do not propagate outward along the "rays" of the Saltire as much as they are evident in their central data profile. While there are logistical rationales for separating out these data models (in that a hypothetical combined standard which encompasses all three would be more complex than existing standards for each in isolation), there are also rationales for integrating these models in contexts such as database persistence/queries and GUI implementations—to avoid code-duplication and the co-existence of multiple similar but autonomous GUI and/or database "modules" in the same application.

Standardization projects such as AIM often fail to consider GUI requirements in any detail, presumably because they are conceived as protocols for sharing information *between* applica-

tions, whereas GUI design has to do with how individual applications interact with their users. Different applications can have different GUI layouts and styles, even if they adhere to the same data standards; in this sense, GUI design (or "visual object" models, using our terms from Chapter 6) is more idiosyncratic, less standardized, across diverse applications than are data models themselves. Similar points could be made about data persistence: strategies for encoding annotations in a relational (or NoSQL) database may be noticeably different than image feature-vectors, for example. This difference can obscure opportunities for applications to have a unified interface for interacting with a database back-end (as opposed to developing data-persistence and query logic separately for annotations and for feature vectors, assuming that there are among the domains which an application seeks to support).

Nonetheless, the trade-offs between complexity and scope are still in effect: wider-scoped data models can help reduce code-duplication in areas such as GUI design and database integration, but engender more complex data models in the core domain. These trade-offs can serve as impediments limiting the scope of data models that become popular as common standards. The desire to keep standards intended for wide adoption relatively simple is understandable, but details of the trade-offs involved may not be fully evident without taking such concerns as GUI design and data-persistence into consideration.

Our proposals for "multi-aspect modular" design may not produce *a priori* solutions to these issues, but they can potentially introduce a framework for exploring software engineering solutions, which find an effective balance among the competing priorities of standards' simplicity and code-reuse. Multi-aspect modules are, by design, relatively self-contained and multi-featured, but distinct modules can share code which is applicable across their respective domains. As such, the combination of modular autonomy and code-reuse provides an infrastructure for optimizing domain-specific code where necessary, but identifying code-sharing strategies when possible.

References

[1] Marie-Laure Boizeau, et al., Automated image analysis of in vitro angiogenesis assay, Journal of Laboratory Automation 18 (2013) 411–415, https://journals.sagepub.com/doi/pdf/10.1177/2211068213495204.

[2] Ethan Bommarito, Michael J. Bommarito, An empirical analysis of the R package ecosystem, https://arxiv.org/pdf/2102.09904.pdf, 2021.

[3] Ayoola Brimmo, et al., 3D printed microfluidic probes, Nature Scientific Reports (2018), https://www.nature.com/articles/s41598-018-29304-x.pdf.

[4] David A. Clunie, DICOM structured reporting, http://www.dclunie.com/pixelmed/DICOMSR.book.pdf.

[5] Chris Cummins, et al., PROGRAML: a graph-based program representation for data flow analysis and compiler optimizations, in: Machine Learning Research, Proceedings, vol. 139, 2021, http://proceedings.mlr.press/v139/cummins21a/cummins21a.pdf.

[6] Andreas Fabri, et al., On the Design of CGAL, the Computational Geometry Algorithms Library, Max Planck Institute, 1998, https://core.ac.uk/download/pdf/210674597.pdf.

[7] Harry Felton, et al., Negligible-cost microfluidic device fabrication using 3D-printed interconnecting channel scaffolds, PLoS ONE (2021), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0245206.

[8] Robert C. Gentleman, et al., Bioconductor: open software development for computational biology and bioinformatics, Genome Biology (2004), https://backend.orbit.dtu.dk/ws/files/4751234/gb-2004-5-10-r80.pdf.

[9] José Guedes da Silva Júnior, et al., Fractal dimension as tool for vascular diagnosis in health, Hematology & Medical Oncology 4 (2019) 1–4, https://www.oatext.com/pdf/HMO-4-187.pdf.

[10] David Juncker, et al., Multipurpose microfluidic probe, Nature Materials 4 (8) (2005) 622–628, https://pubmed.ncbi.nlm.nih.gov/16041377.

[11] Petros Kalendralis, et al., FAIR-compliant clinical, radiomics and DICOM metadata of RIDER, interobserver, Lung1 and head-Neck1 TCIA collections, Medical Physics 47 (11) (2020) 5931–5940, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7754296.

[12] Aditya Kashyap, et al., Quantitative microimmunohistochemistry for the grading of immunostains on tumour tissues, Nature Biomedical Engineering

3 (2019) 478–490, https://www.nature.com/articles/s41551-019-0386-3.
[13] Jens Krinke, Mining control flow graphs for crosscutting concerns, in: 13th Working Conference on Reverse Engineering: IEEE International Astrenet Aspect Analysis (AAA) Workshop, Proceedings, 2006, http://www0.cs.ucl.ac.uk/staff/j.krinke/publications/aaa06.pdf.
[14] Nina Kristine Reitan, et al., Characterization of tumor microvascular structure and permeability: comparison between magnetic resonance imaging and intravital confocal imaging, Journal of Biomedical Optics 15 (3) (2010), https://pubmed.ncbi.nlm.nih.gov/20615006.
[15] Neeraj Kumar, Graph-Theoretic Properties of Control Flow Graphs and Applications, Thesis, University of Waterloo, 2015, https://core.ac.uk/download/pdf/144148533.pdf.
[16] Robert C. Leif, CytometryML with DICOM and FCS, in: SPIE (International Society for Optics and Photonics), Proceedings, 2018, https://spie.org/Publications/Proceedings/Paper/10.1117/12.2295220?SSO=1.
[17] Roland Leißa, et al., A graph-based higher-order intermediate representation, in: IEEE/ACM International Symposium on Code Generation and Optimization, 2015, https://compilers.cs.uni-saarland.de/papers/lkh15_cgo.pdf.
[18] Robert D. Lovchik, et al., Rapid micro-immunohistochemistry, Microsystems & Nanoengineering 6 (2020), https://www.nature.com/articles/s41378-020-00205-2.pdf.
[19] Aaron T.L. Lun, et al., beachmat: bioconductor C++ API for accessing high-throughput biological data from a variety of R matrix types, PLoS Computational Biology 14 (5) (2018), https://pubmed.ncbi.nlm.nih.gov/29723188.
[20] Stephen Machnick, et al., The usefulness of chest CT imaging in patients with suspected or diagnosed COVID-19: a review of literature, CHEST Reviews (2021), https://journal.chestnet.org/article/S0012-3692(21)00689-9/pdf.
[21] Pattanasak Mongkolwat, et al., The national cancer informatics program (NCIP) annotation and image markup (AIM) foundation model, Journal of Digital Imaging 27 (2014) 692–701, https://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC4391072&blobtype=pdf.
[22] Manuel Möller, Saikat Mukherjee, Context-driven ontological annotations in DICOM images: towards semantic PACS, in: International Conference on Health Informatics, Proceedings, 2009, pp. 294–299, https://www.dfki.de/fileadmin/user_upload/import/4088_healthinf2009.pdf.
[23] Animesh Nandi, et al., Anomaly detection using program control flow graph mining from execution logs, in: 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Proceedings, 2016, pp. 215–224, http://www.kdd.org/kdd2016/papers/files/adf1233-nandiA.pdf.
[24] Erika A. O'Donnell, et al., Multiparameter flow cytometry: advances in high resolution analysis, Immune Network 13 (2) (2013) 43–54, https://www.immunenetwork.org/pdf/10.4110/in.2013.13.2.43.
[25] Mariusz Paradowski, et al., Capillary blood vessel tracking using polar coordinates based model identification, in: Computer Recognition Systems, vol. 3, 2009, pp. 499–506, https://link.springer.com/chapter/10.1007/978-3-540-93905-4_59.
[26] Thomas M. Pearce, et al., Integrated microelectrode array and microfluidics for temperature clamp of sensory neurons in culture, Lab on a Chip 5 (2005) 97–101, https://pubs.rsc.org/en/content/articlehtml/2005/lc/b407871c.
[27] Silvius Rus, et al., Scalable array SSA and array data flow analysis, in: International Workshop on Languages and Compilers for Parallel Computing, Proceedings, 2005, pp. 397–412, http://www.csc.lsu.edu/lcpc05/papers/lcpc05-paper-49.pdf.
[28] Mohammad A. Qasaimeh, et al., Microfluidic probes for use in life sciences and medicine, Lab on a Chip (1) (2013), https://pubs.rsc.org/en/content/articlelanding/2013/lc/c2lc40898h, 2013.
[29] Monjoy Saha, et al., AI-driven quantification of ground glass opacities in lungs of COVID-19 patients using computed tomography imaging, medRxiv (2021), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8282108.
[30] Thomas D. Sherman, et al., CancerInSilico: an R/Bioconductor package for combining mathematical and statistical modeling to simulate time course bulk and single cell gene expression data in cancer, PLoS Computational Biology (2019), https://www.math.uwaterloo.ca/~msatrian/papers/cancerinsilico.pdf.
[31] Heshui Shi, et al., Radiological findings from 81 patients with COVID-19 pneumonia in Wuhan, China: a descriptive study, The Lancet (2020), https://www.thelancet.com/pdfs/journals/laninf/PIIS1473-3099(20)30086-4.pdf.
[32] Kenta Shinha, et al., A microfluidic probe integrated device for spatiotemporal 3D chemical stimulation in cells, IEEE Transactions on Visualization and Computer Graphics (2021), https://www.mdpi.com/2072-666X/11/7/691.
[33] Till Sörensen, immunoClust – automated pipeline for population detection in flow cytometry, https://www.bioconductor.org/packages/release/bioc/vignettes/immunoClust/inst/doc/immunoClust.pdf.
[34] Matvey Sprindzuk, et al., Computer-aided image processing of angiogenic histological samples in ovarian

cancer, Journal of Clinical Medicine Research (2009), https://core.ac.uk/download/pdf/8701857.pdf.

[35] Andrew Stone, et al., Automatic determination of may/must set usage in data-flow analysis, in: Eighth IEEE International Working Conference on Source Code Analysis and Manipulation, Proceedings, 2008, http://cgi.cs.arizona.edu/~mstrout/Papers/Papers05-09/scam08.pdf.

[36] Yulei Sui, et al., Flow2Vec: value-flow-based precise code embedding, Proceedings of the ACM on Programming Languages 4 (2020), https://dl.acm.org/doi/pdf/10.1145/3428301.

[37] A. Traverso, et al., FAIR quantitative imaging in oncology: how semantic web and ontologies will support reproducible science, http://ceur-ws.org/Vol-2849/paper-14.pdf.

[38] Johan Van Soest, et al., Towards a semantic PACS: using semantic web technology to represent imaging data, Studies in Health Technology and Informatics 205 (2014) 166–179, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5119276/.

[39] David J. Vining, et al., Development of the ViSion ontology, https://www.semanticscholar.org/paper/Development-of-the-Vision-Ontology.-Vining-Salem/84201e9bd58348bcc5356e0c154a629cc524c778.

[40] Lili Wang, Robert A. Hoffman, et al., Standardization, calibration, and control in flow cytometry, Current Protocols in Cytometry 79 (2017), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=921423.

[41] Edy Wijaya, et al., Surface plasmon resonance-based biosensors: from the development of different SPR structures to novel surface functionalization strategies, Current Opinion in Solid State and Materials Science 15 (5) (2011) 208–224, http://www.ffh.bg.ac.rs/wp-content/uploads/2017/05/Wijaya.pdf.

[42] Clare C. Yu, et al., Physics approaches to the spatial distribution of immune cells in tumors, Reports on Progress in Physics 84 (2) (2021), https://pubmed.ncbi.nlm.nih.gov/33232952.

[43] Yinyin Yuan, Spatial Heterogeneity in the Tumor Microenvironment, Cold Spring Harbor Laboratory Press, 2021, http://perspectivesinmedicine.cshlp.org/content/6/8/a026583.full.pdf.

This page intentionally left blank

CHAPTER

9

Conceptual spaces and scientific data models

OUTLINE

9.1 Introduction

In the two decades since Conceptual Space Theory was first proposed (in a mostly cognitive-linguistic setting), researchers working in a more strictly computational framework have adopted, extended, or formalized Gärdenfors's approach in several different ways. Some have focused on potential AI applications; others on data-modeling and the systematization of scientific data descriptions; and some computational linguists (including via Quantum NLP) have proposed developing a computationally tractable semantics rooted in Conceptual Spaces.

Work such as Coecke *et al.* [14], which we emphasized in Chapter 6, intimates that conceptual spaces offer a path toward resolving the "grounding problem," wherein mechanical systems can never truly have semantics in the first place—there is never a deliberate or conscious correlation of system state to external

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care
https://doi.org/10.1016/B978-0-32-385197-8.00016-7

objects/situations—but merely a simulacrum of semantics engineered to force "internal" and "external" state into a rough alignment, with the effect of endowing AI machines with certain behavioral tendencies that we associate with language-use (and human intelligence in general). Investigations related to "solving" the grounding problem have been among the primary motivations for incorporating conceptual spaces in an AI context (*see* [23], [24], [19], [17], [41], [3], or [4] for good examples). On the other hand, work such as Coecke *et al.* also suggests conceptual space applications to the issue of *construction semantics*, or clarifying how meaning-compositions (which we can also characterize as *concept blends,* so long as there is no confusion with other uses of this phrase, notably Gilles Fauconnier's and Mark Turner as in [20], [21]) such as "dark red," "red square," "wedding ring," "fake diamond," and so forth, signify a compound idea that structurally integrates its parts.

The possibility of finding quantitative dimensions within concept-extensions serves a useful purpose with respect both to concept-grounding and concept-blending. With respect to grounding, dimensional quantification gives us an idea of how concepts may take shape from recurring exposure to certain perceptual or situational patterns, as well as concrete models of concept-learning that could be simulated via neural networks. With respect to concept-blending, dimensional models are obviously applicable to constructions such as *dark red* and *red square,* where concept-blends present intersections or unions of dimensions in straightforward ways (darker hues of red and red-hues plus square-shapes, respectively).

Classifying constructions according to dimensional interactions (e.g., intersection, union, or contrastive) presents a useful semantic outline that applies to many forms of constructions.[1] As we argued in Chapter 6, an equally important construction genre is more exactly modeled by considering concepts playing different roles in larger situations, in which case the *union* between multiple concepts is characterized by concepts being indexed by roles with relatively little inter-dimensional interaction. With that caveat, though, conceptual space models seem to have a solid place in the general system of situational semantics (viz., semantics of situations in general, including but not restricted to information-theoretic situation theory associated with Jon Barwise, John Perry, and Joe Seligman).

The form of syntax/semantics interface pursued by Coecke *et al.* which we reviewed in Chapter 6 has been developed by the same research team in relation to several different semantic paradigms, including statistical word-use correlations that do not purport to offer a bonafide semantics at all (an AI agent which infers properties such as topic-relevance based on statistical correlations across document corpora, without attempting to reason about the "meaning" of any words apart from noting how likely or unlikely they are to appear in a given context, is not "grounding" symbols at all, but rather using probabilities to approximate the behavior of a cognitive system). Coecke *et al.* therefore explicitly state that one motivation for pursuing conceptual spaces as the foundation for their semantic layer is to provide a more cognitive semantics to pair with a grammar based on hypergraph categories. In this sense, their research fits within the "grounding" topic in terms of the methodological rationale for incorporating conceptual spaces. In substance, however, the models they actually formulate—which can

[1] *See* for example [1, page 9]; contrast classes are modifiers which tend to restrict the scope of a conceptual space down to a subregion, as in examples like the "small" of *small planet*, but do not have dimensional structures themselves, because *small* for example can imply many different size-ranges depending on the scale of object to which it is applied (*small planet* vs. *small molecule*).

be glossed as employing a category-theoretic framework to formalize the basic idea that syntactic constructions represented via hypergraph categories can engender semantic constructions analyzed via concept-blending (recall the specific sense of *engender* we adopted in Chapter 6)—are thematically aligned more with construction semantics.

In the context of natural-language theories, to be sure, grounding and constructions can be philosophically interconnected: cognitive construals of integrated situations obviously emerge from cognitive-perceptual discernment of phenomenologically fundamental concepts (so Gregor Strle, for instance, considers the possible role of conceptual spaces as "a mediating level between symbolic and subconceptual representations," albeit less crisp in actual cognition than in the artificially simplified terms of the theory: "methods and models ... are ... explanatory tools, i.e. instruments used to explore, simulate or explain particular aspects of cognition, not mechanisms of human mind." [49, page 18]) As the last quote intimates, neat quantitative models of concept-acquisition/emergence need not be read as realistic simulations of cognitive processes; instead, this like any other theory presents an unrealistically simplified but explanatorily useful model of the actual systems it seeks to clarify. The real conceptual "spaces" in the mind may have many more dimensions, with less obvious numerical encodings, than colors or spatial configurations.

If we accept the simplifying gestures of axis-like concept-dimensions as an imprecise but heuristic gloss on cognitive explanantia, then we can imagine *something like* an emergence of concepts by dynamic clustering of similar experiences (and resolving borderline cases intermediate between related concepts) being among the key driving forces of cognitive symbol-grounding. Likewise we can picture convolution between different concepts' dimensions as defining the contours of the hybrid concepts resulting from semantic blends (and hence semantic constructions, engendered by syntactic concatenations between concept-words). The cognitive dynamics and dimensional structures in both cases are similar (governed by the same conceptual space models), which suggests—even if we assume that real-world cognitive processing is much subtler than such toy models articulate—that there is some cognitive resonance, some redeployment of a common neurological architecture, in the primitive concept-grounding phenomenon as well as the more mentally sophisticated construction-semantics hermeneutic.

Arguably, however, issues of grounding and constructions are more strictly separate when considering the "semantics" of formal data structures, or, in practice, developing working semantic models for resources such as scientific data sets. Certain themes which are central to conceptual space theory clearly have direct applications to computational data modeling, such as dimension/domain structures and region-based classification within and among concepts (for prototypes, concept extensions, borderline-case disambiguation, and so forth). However, rigorous dimensional modeling and geometric clustering are hardly unique to conceptual space theory, and when considering applications of this specific theory to formal data semantics we should attempt to isolate philosophical features endemic to the conceptual-space lineage in itself (as opposed to broadly applicable statistical paradigms). Conceptual space theory's manifestations in the specific natural-language concerns of symbol-grounding and construction-semantics do not necessarily carry over to formal data semantics, or to such practical applications as query-engine implementation or data-set annotation semantics.

In this chapter, by contrast, we will consider the degree to which different use-cases or adoptions of conceptual space theory in a computational setting *can* be synthesized and integrated, focusing attention on practical application within the general scope of *data* seman-

tics. Part of this analysis involves generalizing Coecke *et al.*'s model of the syntax/semantics interface to, potentially, the realm of formal/programming languages, continuing Chapter 6's "Syntagmatic Graph" model. That model provides a framework for codifying the relation between formal-language *syntax* with *semantics* in terms of a notion of "information content" (which we will make more precise). We will argue that this framework presents a foundation for integrating the data-modeling and scientific-computing Conceptual Space use-cases, and in particular that the relevant notions of information content provide an additional formalization of notions central to the original Gärdenfors theory (such as dimensional separability and domain correlation).

9.2 Verb-centric grammars and information-delta paths

Our syntagmatic graph approach assumes a specific reading of Coecke *et al.*'s syntax-semantic interface theory, albeit (we would argue) one which seems well-motivated by their specific text (and moreover is sufficiently general as to be applicable in a variety of contexts): paths in syntagmatic graphs (or analogous syntax-representing structure/spaces) map to "semantic" paths *characterized by an increase in information content*. Each step along a syntagmatic path is mirrored by a correlative step—whose nature is fixed by the underlying syntactic neighborhood (this is precisely the dynamics of grammatical principles; a language's syntactic system should carry enough determinateness that syntagmatic steps, e.g., between word-pairs, impute semantic modifications in a well-determined way)—through a semantic "space" culminating in a sentence's overall meaning (which is the end-point of a semantic path, corresponding to a sentence's root verb as the end-point of a syntactic path).[2]

Semantic paths should follow syntagmatic paths fairly predictably—that is, for a given syntagmatic step, there should not be a multitude of noticeably different semantic steps that are all possible correlates of the syntagmatic step according to the language's syntax-semantics interface (semantic steps that are underdetermined would result in large-scale sentence-level ambiguity). On the other hand, "points" and "steps" in semantic space should not be pictured in too crisp a manner; the full meaning of a sentence only emerges in its entirety, and there may be degrees of detail that remain unresolved while the sentence is unfolding (for instance, postcedent anaphoric resolution, as in *they* to *England* in *although they were playing in London, England lost the 2020 final*). Syntagmatic steps should "compel" paths in semantic space, but not too rigidly; this interplay between syntax and semantics creates a dynamic environment where syntactic norms (and lexical specificity) evolve toward the proper balance of rigor and flexibility. Or at least, this is a plausible working model of the syntax/semantics interface.

Assuming some form of this hypothesis is in effect, we can picture "paths in semantic space" as the *image*, under certain (not-fully crisp) mappings between syntactic and semantic systems, of syntagmatic paths. In our proposed syntagmatic graph model, syntagmatic paths are characterized by the structural tendencies of the underlying graph (in particular they are constrained to lead toward verbs/procedure-nodes except via specific "hinge" nodes or edges), and semantic paths via the general stipulation that movement along the path corresponds with amplification of information content. The overall syntax-semantics interface then ensures (and evolves to satisfy the need for) these two path-

[2]Here we continue the analysis developed in the first section of Chapter 6; we defer to that chapter for definitions of nonstandard terms.

tendencies to operate in parallel. Syntagmatic steps toward verbs/procedures should correlate with amplificatory steps in a semantic space permeated by a measure of information content, and vice-versa.

This picture, though intuitively compelling (we suggest), leaves quite open-ended any specification of what the parameters of "semantic space" actually are. Coecke *et al.* address (what appears to be) an analogous issue by examining the interpenetration of conceptual spaces. It is helpful (even if just as a mental figure) to picture "semantic space" as a space with points and regions, and conceptual space theory legitimates such a picture, because according to this theory many concepts can indeed be given geometric form, in terms of domains and dimensions which possess at least some quantitative characteristics (e.g., the optical mathematics of colors, or the mereotopological interpretations of spatial terms, such as *near*, *over*, *across*, *around*, *inside*, *alongside*, etc.). Coecke *et al.* try to generalize this picture by arguing that (albeit extended to more abstract forms of quantifiability, e.g., the similarity of an object to a prototype) all concepts can be expressed, via suitable choices of dimension, in *some* geometric terms, so we have a basis for modeling concept-combinations in general as unions and intersections between quantifiable spaces. Such generalized spaces therefore provide us with territory on which to define the general notion of "semantic paths," which (even if different terms are used) are the central facets of analyses explaining how syntactic form engenders semantic meaning. Once we have a rigorous model of semantic space, we can develop a theory of semantic paths mirroring (what we are calling) syntagmatic paths.

Since we have argued that paths follow *directions of increase* in *information content*, we will use the generic term "information delta" to designate the degree and qualitative facets of such amplification: *why* does information elevate along a path and *how much*. Clarifying information delta along "paradigmatic paths," or paths in *semantic* space, we claim, furnishes a starting point for explaining how *syntagmatic paths* engender (in the sense of mostly unambiguously determining) semantic constructions. This dynamic can be observed in both natural and programming languages. We will devote the majority of discussion to the latter case, but we will motivate our analysis with a brief review of the natural-language case. This provides us an opportunity to summarize a natural-language theory for syntagmatic graphs that was alluded to (but postponed) in Chapter 6.

Conventional conceptual space theory usually talks about "regions" more than "paths." For example, the "blend" of two concepts represents the intersection of regions where the two concepts share similar dimensions (e.g. *dark red* or *shiny red* represent combinations of color and/or optical properties, which generally share dimensions and therefore yield mixtures that reduce the scope of applicability; *dark red* is more specific than either *red* alone or *dark* alone) as well as the union of dimensions which are not shared (e.g., *red square* defines a concept whose dimensional boundaries include both spatial-configuration and color-optical domains). Implicitly, Coecke *et al.*'s insight is that anywhere we have enough space-like structure (or, say, topos-like structure, since in their context we are working in a categorial framework) to define *regions* (as in the Gärdenfors "concepts are convex regions"), then we also have enough structure to consider *paths*; and whereas concepts are *regions* in that "space," paths are *meanings* in the space, derived from language-like artifacts (structures with a grammar), in the sense that the meaning of a whole corresponds with a path end-point, and the meanings of parts correspond to path segments.

A general notion of "semantic space" accordingly has to serve several theoretical goals: accounts of semantic *regions* should buttress theories of concepts, while *paths* should be analytically associated with "meanings" *under the assumption* of a syntax-semantic interface, where

syntactic rules govern how syntagmatic steps compel semantic steps. Tying these two ideas together, paths in semantic space tend to integrate multiple concepts (correlative to syntagmatic paths visiting tokens that encode those concepts), so *red square* iconifies a step defined via syntax (adjective modifying noun), a step in some semantic space (arriving at the idea of a red square, which can then be plugged in to more general semantic context), and a blend of two concepts (the merged idea carries concepts of both *redness* and *squareness* into the contexts where it is used). Conceptual space theory has tended to gravitate toward blending scenarios, where concept-juxtapositions *narrow the scope* of each concept: *dark red* is narrower than *red* alone, because "dark" filters out some red shades. Likewise *red square* is also a kind of narrowing—even though *red* and *square* do not restrict one another within common dimensions (although their dimensions are not entirely separate, since both involve a minimal sense of spatial *location* even if not spatial *region*)—because *red square* can only be predicated of something with both spatial configuration and optical properties (not *red light*, say). It may be intuitively natural to highlight such "narrowing" effects of concept-combinations due to the overarching theoretical program wherein semantic paths converge to precise meanings: if each successive step combines two concepts, then paths which are chains of such steps evince greater conceptual precision. On that intuition, the examples of conceptual spaces generating narrowing effects from concept-juxtapositions would seem like especially important case studies for the general project of using conceptual spaces to define "semantic" space in general.

This chapter will argue, however, that concept-blend analyses are of only limited value for either natural or formal languages, and will instead focus on issues such as *conceptual roles* and *dimensional analysis*, which will integrate the data-modeling applications of conceptual space theory with the linguistics-oriented themes of the last few paragraphs. In particular, we will continue the discussion of role-indexed multipart relations initiated toward the end of Chapter 6.

We should be careful not to take analogies between natural and programming languages too far: many correspondences between the two forms of languages are valid but superficial, so they do not particularly engender new ideas or technical foundations either for natural linguistics or for programming language design or implementation that would actually advance those disciplines. Nonetheless, even fairly superficial or vaguely specified correlations between theories (or models, perspectives on, and so forth) formal and natural languages can be useful "intuition primes," suggesting new lines of research, which could plausibly yield more rigorous results than the underspecified initial intuitions might intimate. For example, the semantics adopted by Quantum NLP—at least as pursued by the Oxford Quantum Group—arguably bears only a superficial resemblance to conceptual space theory as formulated by Gärdenfors and subsequent linguists, who were working primarily in a cognitive linguistic and largely humanities/philosophical context. (At least, once we consider the kinds of semantics that may actually be modeled according to Quantum logic, i.e., that would currently run on the Oxford group's quantum computing environments.) However, investigating the philosophical resonance between conceptual spaces and the form of semantics endemic to Quantum NLP helps to illustrate—at least if we consider work such as Coecke *et al.* to be persuasive overviews of the relevant technical issues—that conventional formal-language "semantics" (leaving aside questions of "natural" semantics aligned with human pragmatics and conceptualizations) has its own theoretical limitations.

This chapter will consider one specific model or perspective, which cuts across both formal and natural languages, although with the above provisos in effect: we should be wary of plausi-

ble but largely vacuous correlations that might be identified between these two genre of languages, and consider any perspectival arguments or technical models to have merit only if they appear to lead in practically useful directions for either natural linguistics or software language engineering and related fields. We need to try to find ideas of theoretical substance from the larger space of basically trivial summarial notions that are largely self-evident when considering language in general (both formal and natural) and syntax/semantics.

It is obvious, and as such not especially useful to incorporate into a rigorous theory, to note the general point that grammar governs which words are linked with which other words, and that word-pairs have semantic interpretations wherein the pair has a conceptual detail, modification, or specificity that is lacking for either word in isolation. For instance, *beautiful diamond*, *antique diamond*, *fake diamond*, *expensive diamond*, *her diamond*, *some diamond*, and *the diamond* all ground, modify, and/or qualify the generic concept "diamond" in some fashion. This is an example of how word-pairs "compose meanings."

Understanding a sentence—at least according to a wide range of linguistic schools of thought, which tend to leverage this intuition in different ways—is in large part a question of identifying which words to pair with which other words so as to compose meanings "the right way": a sentence is a "composition" built up from word-pairs which are themselves compositions fusing two "meanings." The crux of sentence structure is identifying which words are directly connected to other words to form the building-block "compositions" that are fused together in the larger sentence—e.g., in *the brilliant diamond on her sister's expensive diamond wedding ring* note that *brilliant* is composed with *diamond*, as is *the*; *expensive* is composed with *ring*, as is *wedding* and the second *diamond*; and *her* is composed with *sister*. In natural language, words can pair up according to rules which are not evident in the surface language; for instance, it is not the case that an adjective always immediately precedes a noun (in the above sentence *expensive* modifies *ring* although there are two other words between them; and the possessive links *sister* to *ring* even though the two words are far apart). The fact that meaning-composition may involve words which are superficially distant, and where there may not be explicit markers (such as morphological cues) pointing to correct word-pairings, suggests that syntactic rules serve the specific purpose of establishing which word-pairs are compositionally relevant for any given sentence.[3]

This picture has theoretical ramifications that might not be obvious if we only consider the obvious point that meaning is compositional: that the meaning of a sentence is somehow a product of its component parts. The more specific argument is that compositionality has a certain internal structure, and that we can model composition in terms of *sequences of word-pairs*, where the role of grammar is first to identify which word-pairs are in effect, and second to "rank" the pairs as logically prior or posterior. An open question is how to model the "space" that emerges if we see the ordering of word-pairs as forming a "path"; one criterion of this space should be that paths culminate in something like *propositions* (or "complete ideas"), and also that there is some ambient notion of "information content," such that paths progress *in*

[3]Moreover, because word-distance is not a definitive criterion for the validity or non-validity of sentence-parses in word-pair sets, there is a vast collection of possible parses which are plausible based solely on type-reduction criteria (or whatever combinatory or phrase-structural constraint play roles akin to type-reduction in a pregroup-compatible grammar). This is one reason why Quantum methods may be feasible and yield superior performance to classical computations: the initial goal of NLP is to derive a parse-graph which maximizes "coherence" amongst many parses that are at least *somewhat* coherent. We can, potentially, define coherence in terms of word-pairing by saying that a given (potential) word-pair has greater coherence if the information content yielded by the pair has greater increase on that of each word in isolation than alternative potential pairs.

the direction of greater information content on their way to the "sentence" terminus. A further implication is that sentences have *more* information content (however this is modeled) than their component parts.

This principle is obliquely raised by Coecke *et al*. in the goal of developing a linguistic type system such that "type reductions in the grammar category [map] onto algorithms for composing meanings." Insofar as the *purpose* of grammar, or at least one of its most essential roles, is to govern how addressees decompose a sentence into word-pairs (which semantically "compose meanings"), natural language grammar presumably evolves under pressures to play this role effectively. Parts of speech, for example, can be differentiated in terms of their corresponding roles in word-pair formations; adjectives always modify nouns, for instance, whereas nouns get attached to verbs as subjects and/or objects.

Languages therefore tend to separate different parts of speech through lexical convention (most words' primary meanings are associated with one part of speech in particular) and/or morphological markings (such as modifications which transform lexemes between parts of speech)—in English, for example, the *-ly* suffix converts adjectives to adverbs (*quick* becomes *quickly*). Subtler morphological cues are evinced in case-markings or declensions, which register a noun as subject or object, say (nominative or accusative case, for languages that, more so than English, feature substantial declension markings) or as connected to a verb in some other manner governed by a language's case system: locative, instrumentive, dative, benefactive, and so forth. This general principle of parts-of-speech corresponds, in NLP and computational linguistics, to the idea that lexemes have *types*, and that rules governing when words can be paired up can be modeled via formal type systems (hence the idea of "type reductions"). In short, what we as language users experience as the natural synergy between corresponding parts of speech which engender a semantically meaningful pairing (like *expensive ring*) corresponds formally to a type-reduction rule whereby the collision between two types yields a third type.

This type-reduction phenomenon exists on the scale of individual word-pairs, but a central theory of formal linguistics is that type-reductions can be chained iteratively, which captures the semantic notion that meaning-compositions build up to a "complete idea." For instance, after composing *diamond* and *ring*, we can add a further composition (e.g., *expensive*), or a grounding (*the diamond ring*) or possessive (*her diamond ring*), or supply a verb for which the present meaning supplies a subject (*her diamond ring was an anniversary present*). The *pattern* of meaning-composition is governed by grammar, which provides the essential detail of clarifying the proper *order* of composition—which in turn can subtly (or radically) alter a sentence's overall meaning: *her anniversary present was a diamond ring* connotes something somewhat different, especially in context, than the inverted sentence (*her diamond ring was an anniversary present*); still more noticeably, *she divorced him* is different than *he divorced her*, *she gave him that ring* different from *he gave her that ring*, and so forth. If considered on the basis of their lexical senses alone—i.e., if we did not have syntactic rules as a further source of "information" about speaker intentions—words in a typical sentence can be paired up in many different ways. Syntactic rules therefore need to be sufficiently rigorous that most sentences are decomposable into constituent meaning-compositions (i.e., word-pairs) without undue ambiguity, because they are a key source for clarifying which words are intended to be connected (excluding only word-pairs that make no conceptual sense leaves too many options still available). For this to work, syntax has to be fairly rigorous and rule-bound—arguably implying that it may be

summarized via a formal machinery, such as computational-linguistic type systems.[4]

Coecke's *et al.*'s theory therefore leverages the well-established idea that meaning is compositional, and that this compositionality can be structurally divided into individual word-pairs, where syntactic rules govern which word-pairs are semantically in effect for a given sentence. Their use of category theory to define notions such as "pregroup grammars" largely plays the role of formalizing conditions on how words "pair up": constraints that may be expressed through mathematical formalisms (such as category theory) act as filters—or, more precisely, somehow model cognitive activities that act as filters—which select *some* word-pairs as in effect for a given sentence, screening out others that are lexically plausible, but semantically inaccurate in context (e.g., in *expensive diamond ring* it is, explicitly, the *ring* which is described as expensive, not necessarily implying that there is a single expensive diamond on that ring).

In addition, as well as *filtering* word-pairs, grammatic rules or conventions also *order* pairings so that composition-of-compositions proceeds correctly. It is a foundational principle of cognitive linguistics that the pattern according to which sentence-meanings are built up to a "complete idea" has cognitive significance, even though the contrast between different "compositional paths" is not always logically apparent. For example, *her expensive diamond wedding ring* is presumably expensive *and* hers *and* made of diamonds, so the adjectives string together into a logical aggregate, but there are cognitive reasons why language (or English, at least) seems to compel a fixed order: *diamond wedding expensive her ring* is malformed, even though it has the same adjectives and noun base. Exploration of these cognitive principles is outside the scope of this chapter (*see* for instance Langacker's *Cognitive Grammar: A Basic Introduction* [29, e.g., page 320, or in general Chapters 10–11] for details), but we can observe here that most linguistic constructs demand a fixed word-order, where rearranging the sequence changes the meaning.

Moreover, the order in which words are enunciated propagates to an order among word *pairs*, and therefore among the *compositions* that word-pairs designate. In this sense, the role of grammar is not only to specify which words pair with which, but also to induce a logical ordering amongst all the word-pairs. This appears to be the motivation for Coecke *et al.* turning to conceptual space theory: the logical ordering among word-pairs is a reflection of how meanings compose and, as such, accumulate through a sentence, leading to a complete idea. The *order* in which this composition happens suggests that the emergence of a holistic meaning occurs in stages, so that we can trace a "path" through a kind of conceptual space, where our conceptualization gets more complete and concrete as we cognitively process the sentence in its entirety. Coecke *et al.* use conceptual spaces to model the "path" that captures how information accumulates, how there is an accretion of conceptual detail, which follows the composition of individual word-pairs into a whole sentence.

This chapter will make the further assumption, in the spirit of Cognitive Grammar, that syntagmatic principles can be examined with an emphasis on the *epistemics* of the speaker—i.e., whomever formulates a linguistic artifact—and in particular that sentences' meanings are organized focally around *verbs*. Once again, the cognitive rationales for these assumptions are outside the scope of this chapter, but we follow linguists such as Ronald Langacker, and note that (as cognitive artifact) any verb "profiles" an event, state, or process, and correspondingly, there is for each verb a potential propositional content, or facticity, which when concretized

[4]This is probably a compelling but not self-evident thesis, because there is a possibility that our syntactic instincts allow grammar to play these roles in a manner which is rigorous in the context of cognitive processing, but difficult to formalize in more mechanical environments, such as NLP systems.

produces some sort of "complete idea."[5] For instance, with the generic concept of *moving* there are concrete events wherein something specific moves (and so a specific time and place, or two places, an origin and destination). The cognitive acts which *profile* the given event thereby encompass a specificity that can be expressed in propositional terms. If I see a car moving, say, I see evidence of the fact that the car has moved. Verbs are, as such, intimately linked to propositions in the sense that the concretization of a verb corresponds to the concretization of propositions—note that we could not say the same thing about other parts of speech, such as nouns. The phrase *my neighbor's black dogs*, for instance, concretizes the idea *dogs*, but does not yield a complete/specified proposition.

Earlier we intimated that the progression of sentences toward complete meanings is a matter of "accumulating information," or "accretion of detail"; from this cognitive-linguistic perspective, we can more precisely suggest that such accretion of detail is governed in particular by the tendency of sentences to converge on concrete propositions, and moreover for this convergent process to be guided specifically by the concretization of verbs. In short, the "paths" of sentences through a "space" of increasing information-content can be modeled more rigorously as progressions toward *verb* details: the accretion of information-content is first and foremost a matter of filling in details associated with verbs. At a minimum we pair verbs with a subject, and often a direct object (and sometimes also an indirect objects); we can then add further details, sometimes cued via declensions or related case-markings, specifying data such as *when*, *where*, *why*, *for whom*, *toward where*, and so forth, the verb's event (or process or state) happened or is happening. In short, "paths" in meaning space (however this should be modeled) lead to verbs; word-pairs which do not involve verbs, such as article-noun, are intermediary segments of the path, and ultimately derive their meaning from how the relevant noun connects to a verb, as subject or object (or some more peripheral case-detail, such as location).

This general picture captures (at least as a theoretical hypothesis) the overall cognitive dynamics of language-understanding, where we can visualize the cognitive processes governing sentence-interpretation as an accretion of detail organized around verbs, and the different sorts of relations verbs bear to nouns, which in turn "slot in" to expected roles vis-à-vis the corresponding verb. A ditransitive verb, for instance, presents the "expectation" of a subject-noun, object-noun, and indirect object; whichever nouns play those roles therefore fit "slots" that we perceive as needing specification, once we identify the relevant verb as ditransitive. These nouns are then linked to the verb in networks defined by such expectations, and by the distinct roles played by (e.g.) subject and object vis-à-vis a verb. Such network dynamics fan out from the central verb to other linguistic elements; for instance an adjective modifies a noun, forming one step in the "path" leading to a sentences' "complete idea"; but if the sentence is well-formed this adjective-noun path will ultimately connect to a verb through a *slot* such as subject or direct- (or indirect-) object.

To give a sense of these dynamics with a concrete example, recall Chapter 6's Fig. 6.1 showing a hypothetical parse-graph of a hypothetical sentence elaborating on "went to the store" (there is no special value attached to the precise layout and terms of this graph in the current context, so we won't discuss it in detail). This diagram may convey in pictorial form the general idea of a "verb-centric" grammar and the concomitant semantics, wherein meanings are grounded in the information content they supply to a verb. Tracing paths in such graphs as notated in (Chapter 6) Fig. 6.1 shows how word-pairs (graph edges) lead toward verb-nodes, and

[5] Again, Chapter 11 of *Cognitive Grammar: A Basic Introduction* is a good reference for this branch of Langacker's analysis; or *see* [30], [31], [22], [2], [51], [28], etc.

edges in general are labeled with role-indicators (either subject/object or a declension-case) that summarize the kind of detail supplied by the relevant noun (or noun-phrase) to the target verb.

Such a perspective on natural language makes specific claims about the structural principles which need to be modeled by representations of linguistic form, e.g., descriptions of the parse-trees or parse-graphs of sentences. In particular, we have the proposal that the basic building blocks of parse-structures are word-pairs; that word-pairs can be chained together, and moreover have an overall logical order which retraces a "composition of meanings" at the semantic level; that this ordering is centered on verbs, so that the "chains" among word-pairs (and by extension the words themselves) lead to verbs; that the "links" slotted in to verb represent different roles (subject and object, most notably, but also details provided by different cases according to declension markings, where these are morphological features of the relevant natural language); and that the accretion of detail progresses to something like a determinate propositional content. Such a model can, in essence, be summarized, or formalized, via parse-graphs, where the features just outlined represent criteria on graphs insofar as they model sentences on this paradigm. For instance, such graphs have the feature that their edges can be logically ordered, which in turn would induce orderings between the nodes spanned by an edge (given two edges incident to the same vertex, one edge is prior to the other in the ordering; as such, the non-shared vertex in the prior edge can be ordered prior to the shared vertex, whereas the non-shared vertex in the posterior edge can be ordered as posterior to the shared vertex). Moreover, following these induced edge-orderings yield paths across the graphs, and the idea of semantics as "verb-centric" corresponds to the restriction that all such paths lead to verb-nodes.

9.2.1 The emergent syntax/semantics interface

It might seem that such a theory is still at the vague/underspecified stage without stating more explicitly what "information content" is. That could be, but such a perspective has a further dimension, which may not be immediately obvious; in particular, we are starting to analyze the dynamics that govern *why* grammar comes into effect, or emerges in its explicit form in natural language. Our framing identifies the evolutionary pressures which appear to guide the syntax of language as conventions change over time. In other words, this is a theory not only of the explicit rules we can identify in language grammars but also of the dynamic principles governing the manifestation of grammar as such—specifically that grammar has the role of filtering and ordering word-pairs so as to map sentences onto a kind of "space" endowed with a notion of information content, where meaning-composition corresponds (or "maps") to paths in this space that lead to "complete ideas" (e.g., propositions). Coecke *et al.* capture these mappings in category theoretic terms: "functors" map paths in a space defined by formal *grammars* onto paths in a space defined (to some approximation) by conceptual space theory, with the idea that the former "syntactic" paths somehow guide us (or mathematically model the cognitive processes which guide us) to grasp or follow the corresponding "semantic" paths in "conceptual" space.

Apart from the cognitive and/or computational merits of this theory, it also can potentially lead to new perspectives on the dynamics underlying grammar as such, as just intimated. One way to examine this is to consider the contrast between *syntax*, or the explicit (and to some degree statically analyzable) grammatic rules of a language, with (as it may be called) the *syntagmatics*, or *patterns of lateral organization* observable in language. The term *syntagmatics* is more associated with philosophical linguistics than computational NLP, but in general it

tends to connote an emphasis less on the explicit syntax of language than on the principles guiding the emergence of grammar: the syntagmatic "pole" of language is the order we perceive in how word-sequences follow a consequential progression, so that word-order is neither random nor (in general) freely modified; the sequencing of words conveys meaning no less than the words themselves. The principles governing this ordering are a kind of dynamic arena where specific syntactic rules can be defined, so we address the more "syntagmatic" aspects of language if we investigate the overall dynamics of syntax, as opposed to specific syntactic rules/conventions. Or at least, in Chapter 6, we adopted this kind of usage, and refer to *syntagmatics* as the dynamic principles explaining the cognitive and structural principles guiding the emergence of syntactic rules (where *syntax* as such is less abstractly focused on concrete grammars).

Our hypothesis, to summarize, is that syntagnamic dynamics are driven by the interplay of syntax and semantics vis-à-vis information content: syntactic rules emerge under pressures to engender semantic constructions that embody amplifications of information content in well-structured ways; there is a clear sense of how each part of a construction elevates the information-content as a whole. In the context of conceptual spaces, we would argue that the quantitative dimensions that are internally invoked by particular concepts' cognitive "footprints" are indeed part of such information content—they help articulate how each concept add its own detail to a situational whole—but that conceptual combinations embodied by multi-part constructions should not be modeled (in the usual case) simply by juxtaposing (whether via union or intersection) dimensions internal to every concept spanned by the governing construction. Instead, qualitative dimensions combine with situational roles in potentially complex ways.[6]

This model, we would argue, represents a semantic paradigm general enough to apply both to natural language and formal languages (as well as data semantics). Formal semantics in these contexts needs to consider the internal dimensions of particular "concepts" (or whatever notion plays an analogous role, such as types, or Ontology classes) as well as role-inflected aggregations where multiple concepts are integrated (be this in procedures, multi-part relations, pre-persistent object representations, type-to-visual-object mappings, and other formal metastructures; recall our "Semiotic Saltire" outline). While the terms of such a model are not endemic to conceptual spaces, that theory does serve as a rich starting-point for intuitively driving such a role-oriented picture, because conceptual spaces provide a good case study in how roles as well as intra-conceptual dimensions and details (which present different concept-blending options in constructional contexts) determine "paths" engendered by syntagmatic constructions.

The remainder of this chapter will elaborate on the framework just referenced through various hypergraph-oriented representations of code and/or data structures, including multi-relations, grounded serialization, and type-persistence models.

9.3 Conceptual and thematic roles

Chapter 6's discussion of *multi-relations* (relations with multiple parts and components),

[6]We could also say *thematic relations* as one way to characterize situational roles, adopting terms from Case Grammar and related fields, although outside that specific context the generic term *thematic relation* can take on many information relations, so it may be suboptimal for analyses outside of those targeted directly at case grammar, inflectional syntax, and so forth.

together with distinct *roles* attributed to participants in the relation, alluded to the processes whereby multi-relations contribute multiple facets of information to the knowledge-contexts where they are believed/asserted. Consider an assertion that (continuing that chapter's example) *John* divorced *Jane* (together with details such as divorce and marriage dates). Clearly that relation, to the degree that it is taken and used as fact in some relevant reasoning context, provides specific information. The *nature* of this information depends on our reasoning purposes, which (in a graph-modeling context) amounts to how we are traversing a graph.

If, say, we are "visiting" the John node and wish to learn his ex-wife's name, the "divorce" multi-relation supports that query (by stepping through the relation to the *Jane* node). The multi-relation (at least with data mentioned in Chapter 6) also supports the step of learning a divorce *date* by traversing to *that* (date) node. Similarly, it supports the confirmation that John *has been* married, because there is a path from John to a *marriage* node (via the *divorce* node).

In short, though we speak generically of "information content," we can be more precise about the operations *entailed* by information content by examining the processes afforded by information-encodings (such as via knowledge-graphs). The information content contained within a graph element (e.g., a "divorce" node) corresponds to the traversal-options accorded to paths through or across that element, and how these paths yield data at the sites which they visit. Implicit information content is "accessed"—made *explicit*—by following paths in an information-space/knowledge-space (any space encoding an aggregate body of information/knowledge). Such a notion of *paths* falls quite naturally out of knowledge engineering, and we can consider how it resonates with the analogous picture of *semantic paths* in the context of natural language.

In Chapter 6, we also suggested that multi-relations are in a sense "dual" to *procedures*. Steps in "syntagmatic" graphs involve accretion of detail targeted at supplying enough information to run a procedure. As a precondition for calling a procedure, one must bind values to each required parameter, so all of these parameters point toward the procedure in the sense of supplying data prerequisite for the culminating procedure-call step to proceed. Collectively, each input parameter is then a kind of aggregate step, or a collation of steps into a multi-part step where each step's purpose must be satisfied. Syntagmatic graphs can encode semantic requirements if we argue that procedure-node neighborhoods encode the totality of information which must be available prior to a procedure-call, and that a collection of individual steps, each providing *one* point of info (one parameter-binding), logically entail a kind of *multi*-step which is the epistemic phenomenon of achieving the full information content requisite for a given continuation (e.g., a procedure-call).

The logic of this picture is structurally dual in several different respects for multi-relations in lieu of procedures. A typical multi-relation is traversed from one initial node (one *input*, from the perspective of a specific traversal occasion) and can branch into multiple destinations (multiple outputs); this is dual to procedures taking (in general) multiple inputs to one output. These outputs are possible steps that *may* be followed (so we can picture the steps unified by a disjunction, a modal combination of steps being the set of steps which *may* be taken), whereas procedure inputs *must* be supplied (a step-*conjunction*). In general, a procedure's output for a given input is not known *a priori* (this is why the procedure is called in the first place), so this response is located *in the future* relative to the procedure call; whereas the information contained in a multi-relation has been provided *in the past* (stored in the database/knowledge-base), but now has to be retrieved again. In short, introducing modal and temporal operators serves to sharpen the duality of procedures and multi-relations.

Since duality implies structural similarity, these considerations suggest that syntagmatic graphs are useful for modeling multi-relations as well as procedures, an idea that is borne out by considering how multi-relations aggregate information content. We can adopt the convention that nodes *point in* to a multi-relation node, since each node in the relation's neighborhood supplies data: in the *divorce* example, *John*, *Jane*, *2015*, and *2020* all contribute a field to the data structure embodying their divorce. So we can traverse from the suppliers of data to the target, where that data is synthesized. On the other hand, if we are *traversing* the graph (some time later than when the data is first assembled), we move from the multi-relation node to one of its neighbors (from *John*, say, to *divorce*, and then *Jane*, to learn that Jane was whom John divorced). In short, directions associated with *traversal* are often inversions of those associated with *information content*. This is analogous to the input/output parameter distinction for procedure-neighborhoods. As we argued in Chapter 6, even output nodes *contribute information to* a procedure (e.g., a type-constraint on its return type), so as a *static* model (e.g., for a compiler), a return-value node provides information *to* the procedure-node. However, in the *dynamic* case of actually stepping through a source-code graph, we can step *out* from the procedure to its output node.

In general, the *effects* of a procedure propagate in inverse directions from the accretion of detail which preconditions the procedure. All parameters affected (both input and output) supply information (type-constraints if not actual values) to the procedure, while at the same time the procedure may affect (by overwriting) those parameters. In fact, in type-and-effect-systems one commonly refers to *reading* a value (not just *writing*) as an *effect*, because, rigorously speaking, reading values change the state of their carriers (for example, because the area of contexts where the value is known is then enlarged, which is consequential in logical systems for cybersecurity, say). Thus anything which supplies information content to a procedure may potentially be affected *by* the procedure, with the definition of "affected by" depending on the "effect system" one chooses for modeling procedural side-effects. However, following such effects is not a *traversal step* along graph-paths; rather, it involves tracing evolution in the *program state* which is modeled by source-code graphs. Effect-directions are therefore not explicit structures in the graph, but indirect pathways introduced via code-graphs' semantic interpretation.

In the case of multi-relations a given relation-node centers a neighborhood of multiple other nodes, each supplying some information into an aggregate. The static picture encoded in such a neighborhood involves how those disparate data points are merged into a data structure embodied by the central node (this expresses the aspect of multi-relations where they resemble *objects*). On the other hand, dynamically, a multi-node is traversed *from* a neighboring node *to* a neighboring node (this expresses the aspect where multi-relations act more like "edge tangles"), so "dynamic" traversal represents different path-directions than "static" traversal, similarly to procedures.

The question of *which* node will step "into" the multi-relation, and which will be the destination, depends on dynamic context. For procedures, the totality of information available in the system once calculations are performed is greater than the information statically modeled by code graphs. The graphs capture open-ended information possibilities but closed traversal options, in that one can only "step into" a procedure (using terms canonical to debuggers) after multiple prior steps (for parameter-bindings) are completed, and can only "step out of" the procedure to a specific output node. Dually, for multi-relations, the information is not open-ended (it is fixed ahead of time), but the dynamics of how the multi-relation node is traversed are open-ended in that they are context-

dependent on the state of whatever software is "visiting" the relevant data/information space.

Both procedures and multi-relations reveal structural parallels in terms of static *a priori* data contrasted with dynamic context-specific data, and in terms of how these contrasts align with path-directions on the graphs. In both forms of representation, there is a distinction between explicit graph-structure (which supports certain kinds of paths) and dynamic context, which provides a semantic interpretation to the graph yielding alternative paths (e.g., following procedures to their side-effects). Such indirect paths are not fully explicated in the procedural context, until there is a dynamic context (an actual procedure call), where, for instance, side-effects can be observed. Analogously, paths across a multi-relation to neighboring nodes (the edge-tangle guise of the relation) may only be concretely available with a running context wherein one is visiting a specific input node to the relation. In the case of multi-relations, *roles* can provide guides for these dynamic contexts, consolidating which path *out* of the multi-relation is appropriate for the context.

In effect, it is only through roles that multi-relations can be a basis for well-defined traversal in the first place, because roles specify how to *leave* the relation-node once it is *entered*. For example, the "divorce date" role permits a visitor to step from John to *divorce* and then 2020, to learn the desired datum; that 2020 was when John divorced. Roles provide sufficient structure for multi-relations to be nexus-points from the perspective of graph-traversal.

We will argue that roles can potentially be used in a *procedural* context for in effect a dual purpose, in the sense that *accumulating information content* (adding in data) is dual to *accessing* information content ("pulling out" data). Specifically, roles can play a disambiguating service with respect to how information content *plugs in* to a procedure by analogy to how roles "disambiguate" the range of traversal path-options for multi-relations when their information is accessed. This effect is more pronounced in natural language than programming languages, we will argue, but a similar notion of "roles" can potentially be useful in a programming context.

9.3.1 Disjoint conceptual spaces

Above we suggested that conceptual space theory has perhaps overemphasized the "narrowing" logic of concept-juxtapositions, and implied that we can have rigorous accounts of information content that do not depend on scope-narrowing to realize the paradigm wherein semantic paths converge to precise meanings. Although scope-narrowing is one manifestation of a dynamics tailored to greater precision, what is really at stake in semantic paths—according to the syntax-semantics interface as we outlined it earlier—is that semantic paths lead toward greater information content. In other words, the juxtaposition of two concepts should produce greater information content than either concept alone; however, this model does not require that such amplification occur *because* the concepts' narrow each other's scope.

In this chapter we will turn to *roles* in the AI sense—and indirectly to analogous ideas in Conceptual Role Semantics—to provide an alternative model of information accretion. Two concepts can be juxtaposed to an effect of *more information* insofar as they have *distinct roles*, even if the concepts are not mutually narrowing. In *I drove her to the store* the respective concepts (*I*, *drove*, *her*, *store*) do not appear to combine by limiting each other's scope, but rather by offering different roles to the overall meaning (agentive, patientive, locative). To the degree that we can perceive "spaces" around these respective contexts, it seems that the holistic meaning emerges not from a blending of the respective spaces, but from collating them *in an ordered fashion* by indexing in terms of roles, so to speak.

Conceptual space theory appears motivated by the notion that concept-juxtapositions ac-

quire specificity by how conceptual spaces interact. Word-jumbles, which would seem to tokenize random assortments of disparate concepts (and their spaces) do not carry substantive meaning apparently because they do not permit the concepts to combine in productive ways. This would obviously be the case with pairings having no interpretive path linking one concept to another (*curried theorems*, say). But we can also imagine encountering words in a context where one could suppress the idea that they have any syntactic linkage (consider a scattering of newspaper clippings on the floor). So (say) *red* and *square* taken as just happening to mention two different concepts does not contain any specificity, but the construction *red square* implies that the two concepts are to be *combined*, and this combination is interpreted by intersections and unions of dimensions.

When concepts are combined by attributing them different roles, the pairing is neither completely free-form (they are still a linguistic *construction*, not a random jumble of words) but nor is it dependent on dimension intersection/union for significance. For example, *drove her* is neither an unstructured tokening of two words without any implication that they form a linguistic unit, nor the construction of a fused concept such as *red square*. In effect, roles (subject/object/location/benefactive, and so forth) provide a compositional principle *alternative to* the quantitative models typically advanced by conceptual space theory.

Coecke *et al.*'s strategy, wherein conceptual spaces provide a semantic paradigm above pregroup grammars, for example, applies most directly to simple verb-constructions with subjects and objects. In such contexts—at least for "**SVO**" languages with the verb (for normal clauses)[7] positioned between its subject and object—pregroup structures formally predict the type-reduction of the verb to a proposition, given left-combination and right-combinations with nouns. As with Combinatory Categorial Grammar (*see* [5], [36], or [43]), reductions based on left and right adjacency can thereby formalize conceptual constraints on our semantic interpretations of the relevant clause: reduction of the verb-construction to a proposition captures how our attributing parts of speech to component words guides understanding of how the components should fit together; in the case of a finite clause, they should combine into a propositionally complete idea. Type reductions at the syntactic level thereby map onto communicated meanings at the semantic level, or at least onto interpretive parameters which allow those meanings to be deciphered.

This picture is more complex, however, when we consider the full range of verbs' "theta roles" (which extend beyond subject/object to include patient and agent roles and/or indirect objects), and then the full range of thematic roles available through nested clauses and/or case-marking. With as many as three theta roles and potentially several other thematic roles, verbs do not just "look" left and right for their **S** and **O**; they also accept links to other words or phrases which are neither subject nor object, but instead provide some other kind of detail. All of this complicates the pregroup-grammar model of type reductions (*see* [27, especially Chapter 4] for a formal statement of similar issues).

Additional complications arise from the possibility of "movement" in theta roles, canonically (in English) the transpositions exemplified by sentences such as *I gave the book to John* and *I gave John the book*. Also, *polyvalent* verbs present a problem, because we need to understand which "theta frame" (determining the roles related to agent/patient status we expect to be linked to the verb) applies to each such verb in a given usage; and thematic roles can be "optional" in that we hear some thematic clauses as required by the verb-sense in force, whereas others are supplemental details. The normal sense of *put*, for instance, requires a patient *and* a location (*I put her book on the table*), whereas for, say, *read*, the

[7] Not questions, and so forth.

location is supplemental (*I read her book on the train*).

All of these phrase-structure options force verb-clause theories to have more structure than left- and right-adjoints (as in pregroup grammars) alone, which complicate the calculus of type-reductions over clauses. These may not be intractable problems for approaches such as those of Coecke *et al.* from the *syntactic* side, since multiple thematic roles can always be treated as additional verb-arguments; instead of "left" and "right," we can posit a trivalent or multi-valent set of "directions," which link the verb to different subsidiaries—indexed by thematic role rather than by directions in the surface expression (as an aside, needing to accommodate these multiple "directions" is further motivation for our "syntagmatic" graph paradigm, where all edges point *toward* procedure nodes).[8] Type reduction is then a multistage process that depends on contributions from each component thematically related to the verb.

The question is how to accommodate this more complex notion of type-reduction to Coecke *et al.*'s derivation of semantic constructions being *engendered* by syntactic constructions, with the meaning-combination being effectively an "image" of a mathematically determined map *from* the syntactic construction. The authors show us how this syntax/semantics interface works in the context of simple bivalent verbs, where the relevant semantic "image" is a blend of two concepts. In the case of multivalent verbs, we have multiple concepts which have to be merged together according to thematic roles. And this is precisely the kind of scenario where we have argued that concepts tend to co-exist in situational models with their own internal details, rather than blend into a dimensionally integrated whole. In *I drove the van to Philly*, say, the *van* and *Philly* are situationally autonomous in the sense that one could plausibly drive the former many places, and plausibly travel to Philly via many means. The spaces "around" the patient and location roles are largely disjoint.[9]

In this sort of example, any quantitative dimensions endemic to the respective concepts are in most cases autonomous, as if the sortal effects of their respective roles *inhibits* dimensional convolution. At most, we could say that the dimensions are "latent" and provide cognitive background that can *potentially* interact—an example would be *I took the express train to Philly in under two hours*. Here the geospatial dimension ambient to conceptualizing Philadelphia intersects with the figurative "dimension" involving different kinds of trains, because the sentence intimates that the length of the trip was affected by the train being of the "express" variety. But opportunities for dimensional interactions along these lines are latent rather than intrinsic to role-indexed constructions; in the usual case different concepts (and their dimensions) remain mutually autonomous. To the degree that this is true, quantitative models of concept blending need to be supplanted with situational models that place greater emphasis on how multiple role-construals synthesize into cognitive schemata. It is not clear that this happens on a linguistic level at all, rather than a prelinguistic

[8]Because multivalent verbs imply multiple directions of "adjacency," rather than the binary options of "input" and "output." Edge directions therefore no longer line up neatly with "adjacency" directions; the latter is an additional structural detail *on* the graph-representation, so edge-direction is not interpreted as fixing the imputed direction in this sense. Instead, edge-direction is an artifact of the graph on the basic Syntagmatic level, driven by the convention that procedure-nodes are canonically targets but not sources, which in turn is motivated by the goal of partitioning graphs into neighborhoods with few edges crossing between them.

[9]For the sake of discussion, we assume the role of *van* is *patientive* when it is a direct object (what was driven), though perhaps vis-à-vis cognitive framing its signified role is more instrumentive *except for* cases where the specific purpose of the drive was to deposit the van; otherwise the vehicle is merely the means selected for travel. Syntactically, though, the van is only presented in an instrumentive mode when it is not a direct object (*I drove them to Philly in the van*).

level of cognitive construals and anticipations of the unfolding affairs around us.

Furthermore, insofar as one goal of conceptual spaces is to model similarity and prototype effects, it seems hard to isolate thematic roles from accounts of how we judge the scope and applicability of concepts. In an analysis utilizing *birds* as a case study, for instance, [38, page 158] constructs a "bird space" whose regions correspond to types of birds, analogous to color-concepts being regions in color space. Moreover, there are exemplars within this space such that *robins*, for example, are more prototypical than *penguins* or *ostriches*.

We can, of course, compare birds in different ways (size, color, quickness, etc.), and it is likely that we do indeed as a community share an exemplary bird-notion, which some bird species embody more than others. These two factors make pictures of a Euclidean "bird space" intuitively appealing, because they capture all together the geometric possibility of a prototype, the gradations of similarity/dissimilarity and prototypicality, and the multi-faceted nature of "inter-bird" comparisons. But the success of such a picture in iconifying these intuitions is not *prima facie* evidence that most intra-concept comparisons can be situated along a numeric axis (like size), or even domains of multiple axis-dimensions (like color). Penguins and ostriches are atypical birds largely because they do not fly, and *flight/non-flight* is not a quantifiable domain in any straightforward sense; instead, this distinction depends on our appraising birds' traits and behaviors as functionality organized adaptations to their environment.

Functional criteria along these lines tend to be captured conceptually via thematic roles more than via grade scales. Cars, trains, and buses are all plausible vehicles to take us somewhere but they are not geometrically comparable like red, green, and blue; rather, they provide somewhat different enactive affordances, and each subconcept presents, via its tokens, a distinct genre of functionally organized system. Our background knowledge of how maneuvering into and within a *car* differs from *bus* and *train* is most likely to be exercised in the course of sentences where these concepts would be pressed into service as instrumentives for verbs of geospatial motion. Our cognitive resources for construing such vehicles as "movement tools" supply the scaffolding wherein instrumental constructions along these lines are given sense—what it means for a car/bus/train to be an "instrument." Apparently the same prelinguistic knowledge is manifest in our grasping the similarities and differences between car/bus/train as "vehicle" subtypes. So it seems hard to model the conceptual space wherein the subtypes acquire their patterns of similarity/dissimilarity (and their respective prototypic cores) without focusing on their distinct patterns of functional integration (in terms of how they operate and in terms of how we make use of them).

Though we can abstractly imagine a "space of possible functional organizations" wherein cars, buses, and trains would have their own regions, in reality such a space would seem to be at most an emergent summary of our background knowledge vis-à-vis modes of transport: the basis for our contrasting car/bus/train is all the accumulated knowledge we have of their basic workings, the mental "scripts" we subconsciously follow when entering/boarding a vehicle, or interpreting its movement or planning a trip, and so forth; we do not rely on immediate perceptual cues such as size or color to establish inter-token comparisons, in general. Instead, such comparisons derive from situational reasoning and anticipation as we are either in the process of traveling via car/bus/train or planning to do so.

We do not intend to rigidly pursue this point, since one can find implicit counter-arguments in the conceptual space literature: Gärdenfors and others, for instance, have specifically written about modeling phenomena such as functional organization and such as spatial paths via (quantitative) conceptual spaces [26] (*see also*

[11], [32], [15], [55], [8, especially section 3], or [25, especially pages 5–7]).[10] So there is precedent for modeling (say) verb-plus-locative constructions as indeed quantifiable narrowings (akin to *dark red*), rather than as the basically unblended juxtaposition of two distinct conceptual spaces [56]. Accepting one or another analytic strategy presumably depends on how strongly one is convinced by a Gärdenfors-style encoding of themes such as spatial paths, movements, situational contexts, prototype/borderline contrasts, etc., as quantifiable structures (with sufficient abstractness almost anything can be seen as part of an at least metaphorically quantitative "space," but the degree of abstraction involved may seem to weaken the force of the overarching model, even though we have to grant theories the option of establishing claims in more concrete cases, where the theoretical commitments may be more clearly demonstrated—such as color-spaces as canonical cases of perceptual dimensions—and then generalize to analyses where the terms of the theory need to be applied more abstractly or obliquely). Nevertheless, we can accept the premise that certain cognitive details pertaining to individual concepts have (to varying degrees) quantitative form without arguing that concept *combination*, and therefore meaning-composition (in linguistic contexts) is often defined by a quantitative merger of the two concepts involved.

In *drove to the store* one can, for example, certainly argue that *to the store* has a scope-narrowing effect on *drove*; by itself the verb is open-ended, but the subsequent phrase narrows the verb's scope and concretizes its profiling by supplying a destination for *drove*. This narrowing, however, appears to be a matter of information content being supplied to fill in an abstract slot with concrete details (the "where?"/"to where?" of *drive*); i.e., a narrowing from abstract (compatible with many possible states of affairs) to concrete (empirically specific). This sense of narrowing is different from the refinement effects of, say, *dark red*, and it is hard to see how *to the store* could *quantitatively* alter *drove* akin to how *dark* alters *red*.

In reading Gärdenfors-inspired literature on event semantics one might get the impression that in different places thematic roles (i.e., the concepts that play them) are sometimes understood to *have* quantitative dimensions (e.g., a locative is the culmination of a spatial movement, and so picks up the dimensions of the space wherein such movement occurs) and sometimes understood to *be* dimensions—perhaps reflecting how proponents of this theory have not converged on a paradigm for incorporating event/situational semantics. In, say, *Last week I took some students to Philly by train to attend a conference*, the multiple thematic roles provide a kind of mental "checklist" of details the speaker feels relevant enough to warrant mention. We can picture the full set of (sufficiently relevant) details as a kind of virtual dimension

[10] One problem examined by the authors just cited is that an event construed in different ways could be taken instead—amongst linguists anyhow trying to decide how best to set up philosophical parameters on event semantics—as more than one event, or not. Does *drive to Philly* name the same event as *drive to the conference* in cases where someone does the former so as to do the latter? There are counterfactuals which might suggest that these are in fact two different albeit strongly overlapping events: e.g., were they stuck in traffic and late to the conference; one could still say they were *in Philly* on time. On the other hand, counterfactuals are, indeed, contrary to fact: arguably *any* event *could potentially* have been two events, were some factor to intervene. Event-semantics literature does not seem to settle such issues. But by extension the questions involve seem to impinge on whether spacetime and force-dynamic construals are *sufficient* to characterize events or whether we *also* need some notion of thematic roles, which would help negotiate counterfactuals and other complicating factors. E.g., if we read *Philly* as meaning the *patient* of the drive-to-conference action, then they have not actually driven "to Philly" if they are within that city's municipal boundaries but not at their destination, since in the context of that particular sentence Philly plays a thematic role which is not satisfied by just any trajectory that happens to terminate somewhere in the city. It is not clear how such treatment of the counterfactual would be possible *without* thematic roles, or what a more "quantitative" gloss on the counterfactual would look like.

characterizing the verb (no less than the quantitative dimensions that would literally model the movement, i.e., a spatial trajectory ending in Philly). According to Gärdenfors, indeed, "The cognitive structure of events is relational, gluing together objects, actions and locations" [25, page 6], which sounds as if the theory pictures events as bundling components each of which have distinct cognitive status. The *dimensions* of such a construction would seem to be the components themselves, that provide relata "glued together" into an aggregate. Though not explicitly conceptual space-oriented, but in a theory with detailed analyses of dimensional structures, Lucas Champollion invites a similar reading in summarizing how (in his "strata" theory) "events ... are thought of as occupying regions in an abstract space whose dimensions specify, among others, their spatial and temporal extent" where "thematic roles and measure functions [are also] among the dimensions of this abstract space." [12, page 127] In other words, situations entail a "space of thematic roles" (including but not limited to *theta* roles) and players of individual roles (locative, say) are akin to "points" in this space.

But elsewhere Gärdenfors uses language such as "the force exerted by the agent will be modified by the instrument and thus different from the force vector affecting the patient" [56, page 21], which sounds as if subject and instrument are not *related* so much as "fused" into a "single" force-vector which is the one "affecting the patient." If *that* is the theory's archetype, then thematic roles would paradigmatically blend into quantifiable domains, so the "points" in the blended conceptual space would not be enumerative discretized spaces of possible thematic roles but would be more mathematical and quantitative, e.g., force-vectors, which seems a different picture than "objects, actions and locations" "glued together" (at least insofar as the "glue" is a situational inter-coherence innate to thematic roles).

Our point is not that Gärdenfors is being inconsistent, but rather that neither thematic relations nor quantitative dimensions alone can model event-semantics: roles and dimensions situationally interoperate in many different ways. For example, Champollion's strata theory, which we just mentioned, is a case study in how formal semantics can integrate the qualitative aspects of thematic roles with the quantitative features of conceptual dimensions characterizing the concept-instances which play those roles.[11]

In general, we would argue that roles more often than not *inhibit dimensional interactions*, so that quantitative representations such as Gärdenfors's "two-vector" system typically apply

[11] Dimensional structures help to organize semantic constructions in a number of patterns (such as partitives, mereologically inflected reference, and quantifier-scope): attributes such as mereology and granularity (as well as quantitative notions of magnitude and comparison operators) can all play a role in the semantic rules governing part/whole relations (in contexts such as *each of the ...* or *a few of the ...*), gradations (*more ... than ...*), effects related to the completion or partiality of events (*for an hour* vs. *in an hour*), and so forth. All of this builds up a theory of dimensional structures as constituents of referential scenarios whose linguistic encodings appear to be conventionalized within construction templates involving partitives, quantifiers, comparatives, etc. However, Champollion explicitly allows for these concerns to be applicable to different thematic roles within an overall event semantics: to the degree that (glossing his arguments with terms other that his more precise, but technical alternatives) *dimensional structures* offer a kind of *semantic frame* governing recurring quantity/mereology-related construction-patterns, then such a frame "distributes over" (which *is* his term) thematic roles. Intuitively, each player of a role (each "theta," in his terms) can have its own such semantic frame. Dimensional structures are, that is, "θ-indexed" in the sense that θ becomes one "parameter" (selecting the relevant thematic role) alongside parameters defining dimension-related structures such as granularity (*see* [10, page 33]). This would be an example of how dimensional structures come into play *in the scope of* one specific thematic role, so that the totality of dimensions appertaining to a semantic construction—and to the semantics of an event—can be defined only by considering *both* the qualitative distinctions between roles *and* the quantitative aspects of concepts through which we make sense of the entities playing those roles.

only to agent and patient roles; other thematic roles should be analyzed situationally, rather than as "vectors" (although they may have latent dimensions that can *potentially* be semantically convoluted with the central agent/patient schema in some cases, as in our express-train example, which availability should be taken as part of the semantic arsenal comprising a clause's overall meaning).

For the most overarching analysis (such details as "latent" comparatives aside), rather than treating concept-combinations as meaning-effects which have to be modeled numerically, we can instead interpret concept-constructions as organized situationally: different concepts lend different sorts of detail, each of which add concreteness and specificity, leading from situational abstractness or prototypes to concrete states of affairs. Ordering concepts by their conceptual *roles* can thereby take the place of quantitative modeling for conceptual blends. This role-inflectional approach—which we might characterize in terms of *qualitative* rather than *quantitative* concept-combination—is arguably better motivated by natural language. The fundamental aspect of qualitative combination in this sense is that the spaces of two concepts typically do *not* blend but rather remain intellectually isolated, with the rationale of the concept combination being articulated through distinct syntactic and (correlatively) situational roles rather than through dimensional interconnections. Dimensional structures for individual concepts may still be important as part of our holistic semantic assessments, but in many contexts the situational composition fusing multiple concepts is more significant for overall meaning than the precision afforded by concepts' quantitative dimensions (e.g., the color-space boundaries marked by the concept *red*).

Autonomous concept-blends still, we would argue, evince certain (often partly metrizable) dimensions; indeed, concepts "glued together" have a larger space of domains and dimensions than concepts individually. Given a kernel event like *travel to Philly* we can elaborate on many dimensions detailing that situation, such as *how long*, or *where exactly*, or *when*; some of these details take a scalar or geometric form. But here we transition from dimensional structures *constituting* constructional meanings (as is arguably the case for simple aggregates like *red square*) to dimensions being *available for elaboration*, which implies that they supplement a completed signifying process, which the dimensions themselves (or any interactions and convolutions thereof) do not fully explicate.

Indeed, to the degree that concepts within a construction are autonomous, it is not immediately evident what there is to analyze—we can always say that language-users "glue" concepts together in the mind, and that they are cued about concepts' respective roles by morphosyntax (which so to speak "sieves" concepts into distinct thematic roles), but that sounds like simply restating the explanandum rather than explaining it. The charge for the *syntactic* side of the theory is clearer, because linguists have to demonstrate how clauses are parsed into role-attributions in the first place. We claim that a thorough account of such a process can be derived via expanding from pregroup grammars to hypergraphs, where graph edges track (collections of) nodes to corresponding verbs via the roles they carry.

One can indeed find apparent rules or conventions in clause-structure which appear to govern this process, such as patterns in how thematic roles become registered via theta-roles or relegated to secondary status, and the relation of roles to speaker epistemics. Such observations can found a bonafide "theory" of clausal construction on the *syntactic* side. But semantically, the more that concepts are distinguished in a construction by playing distinct roles, the harder it is to find convincing formulae reducing their aggregation to some formalizable meaning-generator, as opposed to invoking a largely prelinguistic "situational cognition" outside the scope of (linguistic) analysis.

We contend that approaching semantic constructions from the perspective of information-delta *paths* sheds some light on this situation, and can form the core of a theory which does more than hand-wave at the problems of role-indexed semantic constructions. Even within such a theory, nonetheless, we should be receptive to the possibility that unpacking semantic constructions really *is* in many ways pre-linguistic; if nothing else, a theory can demarcate the boundary between processes endemic to language-understanding—as they play out in the context of semantic constructions—and those which defer to "situational cognition," something perhaps too subtle for linguistic analysis and arguably too "human" for "artificial" intelligence.

Yet what we *can* do, even in the confines of linguistics proper, is identify where patterns in situational reasoning (including intersubjective "theory of other minds" effects) appear to drive linguistic (and pragmatic/discursive) conventions, for example in how clause-constructions are organized around the "speaker's point of view," and the need to establish common reference points and situational framings between all participants in a linguistic context. That still leaves a lot for linguistics to talk about. Even if some majority of cognitive situational process is pre-linguistic, there are still many different mental operations which could be pressed into service to construe the affairs around us. Language surely uses semantic and syntactic cues to activate certain prelinguistic capabilities from the full set that could be potentially triggered within ambient situations.[12]

A perspective which emphasizes concepts' situational roles can thus leverage many facets of conceptual space theory, even while de-emphasizing mathematical accounts of concept-blending, and moreover would still be consistent with data-modeling and meta-scientific applications of conceptual spaces as found in projects such as Conceptual Space Markup Language. That is, a *conceptual role* based framework can be one way to integrate this data-modeling branch of conceptual space theory with the formal-linguistic considerations prioritized by (e.g.) Coecke *et al.*.[13] It is in this guise, one can argue, that conceptual space theory as a model for *natural language* semantics can also be a reference-point for settings such as data and code semantics, which lack the former's situational nuance.

There is, however, one useful analogy which may be drawn between the natural-language and data/code semantics cases. Consider the basic idea that role-indexed constructions tend by default to allow component concepts' relative autonomy (by contrast to conceptual blends without obvious thematic-role decompositions, such as *red square* or *dark red*). Due to such autonomy the ultimate "meaning" of concepts' combinations might not be constructed through linguistic means at all, but rather defer to cognitive-situational faculties. The relevant analogy in data semantics is that the full semantic constitution of a data space—of the types and schemas which are embodied by, instantiated in, or constraints on a data set or database—cannot be defined through schematic declarations or axioms alone (e.g., Web Ontologies). Data sets' full semantic import derives from how the software components that use them replicate or analyze some real-world situations, which ultimately depends on *procedures implemented* more

[12]Recurring patterns in situational construals, perhaps especially *perceptual* construals, can then be seen to engender specific types of "semantic frames" (for force dynamics, mereology-related as mentioned above vis-à-vis Champollion, modality (in the sense of modal logic) and counterfactuals, belief-attributions and "other minds," etc.) which encapsulate how language via construction-templates and patterns can invoke cognitive faculties, not just isolated mental processes but interconnected sets of logical and operational schemata through which we organize conscious experience.

[13]*See*, e.g., [39], [35], [45], or [42] for overviews of "Conceptual Role Semantics" in particular.

than on static representations, such as data fields or inter-object relations. Thus data modeling is only preliminary to code implementation, and the *semantics* of data models lies largely in how they can guide, and then make reference to (as data-structural specifications) implementation features and procedures. We would argue that this is roughly analogous to how features of linguistic constructions both *refer* and *defer* to "cognitive procedures" which provide our ambient situational reasoning prerequisite for language.

9.3.2 Conceptual spaces and scientific data

When multiple concepts are merged into meaningful constructions, each concept contributes distinct information which collectively produce a larger information content. Without referring to information content specifically, conceptual space theory alludes to this process and presents one version of how it operates, particularly in the context of conceptual blends: blended concepts (such as *dark red* or *red square*) quantitatively mix their concept-components to procure a more narrowly extended (and thus more information-bearing) prototype than each concept on its own. As we have argued, however, we can continue the general idea of information-content amplification without restricting analyses to these blend-cases; for example, concepts combined qualitatively assemble information content by organizing concepts' contributions in terms of situational roles more than interpenetrations of dimensional structures. Yet however we figure concept-combination, the crucial detail is marking how such combinations use the coordination of multiple concepts to augment levels of information content against that of concepts individually. This foundational question appears to lie at the core of conceptual space theory as a linguistic semantics.

Not long after Gärdenfors's initial language-focused publications, there emerged some follow-up research which shifted emphasis from the semantics of natural language to that of scientific theories and scientific data. This included formulation of Conceptual Space Markup Language (CSML) as a data-description framework, with an emphasis on conceptually and statistically rigorous documentation of the parameters that collectively define a scientific theory or model. The CSML language is a rigorous and well-motivated approach, which deserves to be widely adopted in some fashion, particularly given the more recent emphasis on data sharing and research transparency—which is even more pronounced now than when CSML was first published—as well as the presence of more recent data-sharing protocols that could be integrated with CSML, such as the Digital Curation Center (DCC) lifecycle [48], [50], SciXML [16], [44], [46], [33], [53], [54], IeXML [37], [40], [47], the suite of MIBBI (minimum information for biological and biomedical investigations) guidelines [52], [34], and Stuart Chalk's **SciData** Ontologies [9].

With that said, although formalizations of conceptual spaces such as CSML fit in well with these various attempts to standardize scientific data-sharing, much of CSML's details or vocabulary is not endemic to conceptual space theory *per se*. For example, conceptual space theory places particular emphasis on concepts' (and by extension scientific models') domain and dimensional properties, which in the data-modeling context inspires especially rigorous attention to phenomena such as the statistical scale (nominal, ordinal, interval, ratio), measurement units, ranges, and autonomy/correlation among different modeling parameters. However, documenting such statistical qualities of a data set is hardly specific to conceptual space theory.

On the other hand, notions such as "contrast classes" and the partitioning of a larger conceptual space into regions associated with concept-prototypes *are* more specifically rooted in Gärdenfors's original theory, but such techniques seem more applicable for use-cases such as NLP-driven text mining of corpora *about* scien-

tific models than formulating (digital representations of) models in the first place.

In short, the quantitative techniques derived from conceptual space notions of concepts' dimensional overlap or prototyping are more relevant for mining data sets already established than for constructing and sharing data sets in the first place. These are the same quantitative methods which we earlier qualified vis-à-vis applicability to natural language: we argued that there are many cases of concept-combination that do *not* involve (at least to a substantial effect) numerically analyzable dimensional blends, and that role-based situational models can take the place of quantitative inter-dimensional calculations. It seems plausible that role-based models could offer a similar perspective on concept-blending in the area of data models.

When formulating a data model, the core project is to identify the data types through which information contained in (instances of) the model may be classified, to define those data types in terms of the specific data points and fields they aggregate. The key organizing principle is data *fields* which aggregate into data structures, or compound values, that are type-instances (which therefore have internal structure, but also some integrity as a single conceptual unit). Data fields can in turn be single or multi-valued, with multi-valued fields typically being variant-sized collections such as lists or unordered sets. A specific data point is attached to a type-instance through the value of an individual (single) data field or one value inside a collection (multi-value) field. Via such connections, each data point contributes some information to the total data encompassed by the larger type-instance. In effect, type-instances encapsulate the totality of information content contained in their component parts.

None of this discussion is particularly original or insightful, but it is worth highlighting these basic principles so as to draw attention to the implicit role of *information content*: each field within a type-instance amplifies the larger instance's total information content, which is the basic rationale for joining fields to instances. Data models are successful to the degree that they reinforce this phenomenon. For instance, metadata associated with specific data fields is valuable if it *increases the degree to which* the field-value augments the information content of the larger object. Annotating fields with scale and metric info (referring back to Chapter 6, Section 6.3.1) provides benefit, because, at least in many contexts, such annotations increase the amount of information supplied by an individual data field.

This perspective points to a strategy for integrating conceptual space theory more rigorously with data-modeling paradigms: we can adopt conceptual spaces to study the degree and nature of information-content elevation that is associated with individual data fields seen as information-contributors, whose contributions are in turn mediated (and potentially augmented) by the relevant data model. Conceptual space theory, as we have argued, tends to analyze *increase in information content* via quantitative models of conceptual-blends, but we have suggested that such quantitative accounts are merely one way to operationalize the basic idea of studying information-amplification as a construction-driven phenomenon. In the data-modeling context, we can similarly focus on the more general rise in information-content, which may or may not be driven by numerically analyzable combinations between data-contributing elements. What we can focus on instead is the core principle that data *models* add value to data *sets* by *increasing the degree to which data-contributing elements in each model-instance augment the information content of the overall model* (for the sake of discussion, we can call these *information delta* effects). When considering strategies for developing or refining data models, then, the essential question to ask is *how does a particular strategy contribute to the models' effect of raising the information-amplification of individual*

elements? This points to some tactics for defining, assessing, and formalizing data-modeling strategies: to rigorously characterize a strategy is effectively to clarify how that strategy contributes to information-delta effects.

Likewise, defining delta effects can lay a foundation for strategy implementations, in terms of kernel operations for query-evaluation virtual machines, for example, which would allow data-validation and data-integration strategies to be implemented as sequences of kernel operations. That is to say, a theory of delta-effects should guide construction of virtual machine operation-sets in the data set and data-query contexts. In this sense, modeling information-delta vis-à-vis data sets serves as a continuation on the related notion of information-delta in the context of procedural code models discussed in Chapter 6: in both cases information-delta analysis can be practically operationalized via query-evaluation virtual machines.

9.4 Delta roles and conceptual space markup language

Recall that in Chapter 6, we highlighted "selection" and "instantiation" concerns in the context of queries against data sets. To reiterate, imagine a data set (or alternatively a database) as encoding the totality of its information in the form of a single graph, so that pulling information from the data set is analogous to performing graph queries. Assuming the data set is strongly typed, one property of such a graph is that all of its information content is sorted into type-instances. In general, this means that structures within the graph—which may be hypernodes, edges, properties, and so forth—supply data-points that define a specific type-instance (either the values of data-fields indexed by name, or one value in a multi-value collection).

Data spaces can of course have many different structures; only in special circumstances will they be *explicitly* encoded via graphs. With a suitably general and expressive graph model, however—for instance, one which combines property and hypergraphs (as discussed in earlier chapters)—we can always treat existing data-set layout as isomorphic to a graph schema subject to certain evolutionary and usage/query constraints. For this reason we will proceed by anchoring all discussion in hypothetical data sets presented as hypergraphs, setting aside the details of how to translate queries formulated in other context to queries against hypergraphs.

9.4.1 Information delta and data modeling

Starting from the principle that certain graph-sites anchor type-initialization opportunities, we can refer to an *instantiation neighborhood* of a graph, demarcated by all parts of the neighborhood supplying data to the same type-instances. For example, a type-instance may be encoded via one hypernode, leveraging values inside the hypernode, along with (potentially) properties asserted on the hypernode itself or any node inside it (which may carry other data-points). We can then expand the neighborhood to include other hypernodes, which are necessary to initialize the neighborhood's "root" instance. In this sense neighborhoods may overlap.[14]

According to the terminological conventions we employ here, a "site" in a graph is any structuring element: hypernodes, "hyponodes" (nodes inside other nodes), edges, properties, channels, named subgraphs, and so forth. Sites are associated with type-instances based on the

[14]If desired, we can distinguish hypernodes whose purpose is to provide the equivalent of individual data fields—e.g., multi-value collections—from top-level hypernodes that carry instances of types which are primary from the point of view of the governing data model; secondary type-instances are then logically "part of" primary type-instances, providing at least in terms of logical interpretation a "nesting" effect characteristic of hypergraphs.

neighborhood where they are contained.[15] Each site then contributes some data, directly or indirectly, to its associated type-instance. Any query engine targeting the graph must therefore be able to identify *how* sites contribute data, and incorporate these details into its query-evaluation strategies, particularly in the context of selection and instantiation queries.

To demonstrate, first consider "instantiation" or "initialization" queries, which recall are those confirming that in the neighborhood of a given graph-site we have sufficient data to populate an instance of some type. Essentially, this means at least that there are data points for every field that is required for initializing a value of that type, where these data points are embodied in structures attached to the site. Determining instantiation-queries therefore involves matching requisite fields against sites in the relevant neighborhood providing those fields' data. For such a process to proceed unambiguously, neighborhood-sites would be annotated with metadata specifying how they contribute to instance-initialization for some type; this metadata may be inferred by the graph engine or directly asserted by the governing data model. If it is inferred, we assume that the basis for this inference is some unambiguous property of the data model, with the data modeling language being formulated at least in part to drive that kind of inference. For instance, associating subvalue-nodes (hyponodes) with a named field and declaring that field as necessary for type-initialization—as part of the declaration for some type—signals that a subvalue indexed via that name is an initialization-precondition, and that the relevant hyponode thereby plays an initializing role in the surrounding neighborhood. The relevant type- and field-documentation therefore serves as a form of initialization-annotation.

[15]Without further qualification we will use "neighborhood" in this discussion to loosely mean "instantiation" neighborhoods (in our terms; note that these are somewhat different now from the definitions we proposed in Chapter 6).

In effect, then, we can stipulate, as a precondition on a suitable data-modeling framework, that annotations may define how data sites contribute to type-instantiation (allowing for this metadata to be implicit in other, related declarations). Considered from the perspective of *information content*, we can then investigate how each site's contributions to an initialization (the nature of the information they supply) augment information content in different ways.

For example, suppose we have either a property or a subvalue-field providing a named field (in a record-like data structure), and *moreover* an annotation asserting units of measurement. This could take the form of a global guarantee that values for this field will always be represented according to a specific measurement-scale, or alternatively a contract that such units-data is available as a component data-point in the scale-delimited values on a case-by-case basis. These annotations therefore provide information about units of measurement which affects how the relevant values may be used to initialize neighborhood type-instances.

It is conceivable that scale-annotations are not necessary in some contexts because of alignment between data sources and data-processing routines which ensures that measurement details are fixed and unambiguous for the lifetime of the data set, but in general the presence of scale-annotations would supply a greater quantity of information than raw values without such annotation. If, under these motivations, a data model explicitly requires that units declarations (for any quantities that are not simple magnitudes) be confirmed as a precondition for type-initialization, then that facet of information content supplied by the annotated site fits in to the type-instantiation concern; it is relevant to instantiation queries and forms part of the interface implicit in applicable data sets for constructing values of the corresponding type.

On the other hand, if units-annotations are not requisite for *instantiation*, they may still be important for *selection* queries, if ranges in the

corresponding data type are part of query criteria that would filter potential type-instances. When (say) querying a bioimage database for diameters of regions-of-interest, it is necessary to be sure that one is comparing regions by the proper scale (e.g., centimeters, or percentage of image-width) against the query range.

In any case, the information provided by a given graph-site can be classified into different facets, which we may call *roles* (establishing a connection to multi-relations as discussed above), and these facets can become relevant for different varieties of queries. The *type* of data represented at a site, along with its actual raw value, correspond to two different roles: for initialization queries (if we only want to ascertain that initialization is possible in some neighborhood, which is different from *constructing* the type-instance) the *type* is almost certainly relevant, whereas raw values are not; but raw values would come into play for selection-related queries.

Metadata such as units of measurement could potentially be associated with type-declarations, as a kind of further contract annotating the type, and come into play for questions about whether instantiation is possible. Conversely, scale-declarations may not be directly examined until raw data is actually compared against some range. But in either case the data model expresses the *sorts* of queries where metadata characterizing the site information's *role* becomes relevant. As for scale-delimited values, similar comments would apply to other metadata properties associated with data fields, such as valid ranges, or expected distributions (quantifying how much a fixed value for the field is typical or atypical, to the degree that this can be asserted of a single field in isolation from others in its enclosing type), or such as similarity measures (quantifying the degree to which differences in value on some axis, or the lack thereof, contribute to dissimilarity or similarity between two type-instances).

Insofar as each site contributes to some *increase* in a data set's overall information content, these *roles* serve to define this "delta" effect more granularly. For this reason, we will refer to such roles more precisely as *delta* roles, with the idea that for each delta role there is a specific mechanism through which some data or meta-data adds information content: as a type attribution, a raw value, a scale (or range/distribution/distance-metric etc.) annotation, and so forth. Delta roles can then combine with assertions of query *contexts* where the role-specific information comes into play (e.g., type-initialization vs. instance-selection).

We propose these meta-data annotations being implemented as explicit data-modeling features, rather than left implicit in database schema. These annotations could then be directly exploited by query-evalution virtual machines; they might be built in to the virtual machine architecture and operation-set to a degree that would be infeasible without an explicit accommodation for such metadata in the data-modeling paradigm. Fig. 9.1 outlines these ideas through (what we'll call) a "localized" syntagmatic graph, diagramming relations among sites in a hybrid property-hypergraph via a Syntagmatic representation. Delta roles can also aid in traversal implementations; for example, checking treatments of scale-units (recall Chapter 6's discussion of tracing program flows relative to scale-sensitive procedure inputs/outputs) could proceed by following paths determined by graph-sites annotated with declarations that scale-unit meta-data is in effect.

A formal elaboration of role-delta information would overlap significantly with conceptual space implementations, particularly as these are applied to data-modeling (notably CSML); as such, we incorporate CSML into the query system considered here for data integration specifically as a component of the mechanism for defining and using delta-roles. We will explain this tactic

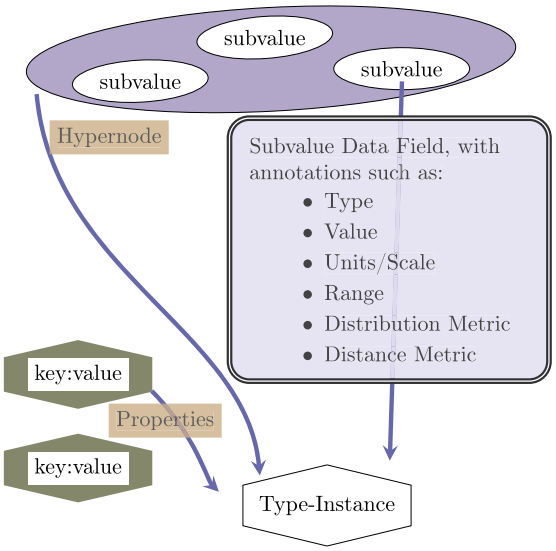


FIGURE 9.1 Role-deltas on a localized syntagmatic graph.

for incorporating CSML with reference to sample data-sets published in this book's supplemental materials.

9.4.2 The artificiality of data semantics

One challenge facing any theory which attempts to mix ideas from both (on the one hand) natural-language semantics and (on the other) contexts such as data modeling and/or programming languages is that semantics in the former sense exists in cognitive and conceptual surroundings that do not exist in the latter sense. Natural language is spoken by people, in (typically) enactive, interpersonal, goal-directed circumstances, where the ambient situation provides frames of reference that prime certain conceptual foregrounds and backgrounds. Language is not suspended in a pure logical space, but rather supplements each person's prelinguistic comportment to the scenarios wherein linguistic activity takes place.

This is one sense (which we will not explore further here, but would be worthy of detailed study in a more philosophical context) that linguistic meaning should be conceived in "delta" terms, i.e., in terms of information-increase: any meaning expressed in language *adds* content to the totality of conversants' cognitions already believed, perceived, experienced, or posited as an object of thought. Many of the structures we might use to characterize languages—syntagmatic constructions involving thematic roles, situational context, and speaker-relative epistemics, for example—also apply to cognitive attitudes that are equally present in the absence of (at least explicit) language (allowing for the possibility that language-acquisition *in general* restructures cognitive dispositions in ways that leave an impression in how people perceive the world around them, so that language is always somehow virtually present; we speak here only of *concrete* linguistic content). Any linguistic entity or pattern need merely trigger prelinguistic situation-comprehension faculties to play its relevant constructional role.

None of this prelinguistic background is especially relevant for the semantics of digital data and computer code, except for AI projects that try to bridge the gap between *in silico* calculations and human cognitions. The issues of symbol-grounding and construction semantics, which we touched on at the beginning of this chapter, are preconditions for a realistic AI simulation of language competence, and embodies lines of research connecting conceptual space theory to computational models. In this section, however, we are concerned with more prosaic data and code modeling scenarios. Since in such contexts there is no ambient prelinguistic consciousness that can ground semantics in cognitive (situational/perceptual) dispositions, a minimal requirement for discussing data/code semantics is what such semantics *is*. Articulating what exactly falls under the scope of "semantics" in these artificially formal contexts helps to

clarify the intuitions behind particular authors' semantic models.

We have already defined the basic elements of our specific construal of data/code semantics, so at this point we need merely connect them to this underlying topic. Insofar as *data semantics* involves the useful structuring, encoding, serialization, and interface-design for data sets and data bases—particularly data sets or database schema that can be annotated with metadata clarifying the semantic constraints respected in the overall data-curation process—the core elements of data semantics are embodied in *instantiation queries* and *selection queries* (here we will define "initialization" queries as a follow-up to instantiation queries, thus establishing the technical distinction between these terms). To reiterate (see Section 6.3 for our preliminary discussion): instantiation queries are, then, concerned with whether a data-type instance can be initialized from a given site in a data set; initialization queries are those used to populate data fields necessary to actually perform such instantiation; and selection queries ascertain whether a type-instance would meet given criteria (were it to be instantiated). Data semantics should also recognize the possibility of "fiat" instantiation, where queries against type-instances can be evaluated without actually performing an instantiation-step if there is some other way to pull the relevant information from a data set/base.

Alongside data-semantics, whose key phenomena are "queries" of the forms just mentioned—or algorithms/capabilities which are functionally similar to such queries—we would also emphasize *code semantics*, whose essential elements are mapping symbols (or graph-nodes) to procedures (procedure resolution), and dynamically calling or examining procedures with parameters populated from data-structure (e.g., graph) sites (call this *procedure reflection*). In this context, we will allow the term *reflection* to include static as well as dynamic analysis of procedure metadata, such as finding pre- and post-conditions or selecting which is the best procedure to call from a group of plausible candidates. Such reflective capabilities would certainly apply to the process of resolving runtime-reflection calls, but we will more generally speak of "reflection" as any process for acquiring and/or incorporating procedural metadata at one or more stages of software engineering, including compilation and runtime as well as pre-compilation static analysis, requirements engineering, design and testing, and so on.

In any case, considering both data and code, the core elements of data/code semantics together are *instantiation queries*, *initialization queries*, *selection queries*, *procedure resolution*, and *procedure reflection*. We claim that almost all "semantic" issues in computational contexts (again excluding NLP-inflected AI) can be classified into one of these five concerns. This assertion, to the degree that it holds up, can propagate to the design of (code and data) annotation schemes and query expression/evaluation systems. For each element of an annotation or query expression, we should be able to identify which of the five core code/data semantic concerns it targets. Likewise, query evaluation systems could incorporate this code/data semantic model into their implementation architectures: primitive query-engine operations can be grouped according to which of the five concerns *they* target. We propose to employ this overall architecture to formulate a query and serialization language, as well as a query-engine virtual machine, which would incorporate precedent technologies such as Conceptual Space Markup Language and the Text-as-Graph Markup Language (TagML) [7], [6], [18].

9.5 Conclusion: Toward a scientific data semantics

When conceptual space theory has been applied to technology—stepping outside its philosophical/linguistic origins—one of the central

goals has been to develop a more useful and realistic semantic model for data structures (more or less what we here refer to as "data semantics"). This is certainly evident in explicit comparisons against the core technologies of the Semantic Web—Ontologies, Resource Description Format (RDF), and so forth—and the outright argument that "the Semantic Web is not very semantic" [24, page 2]. Here Gärdenfors implicitly critiques the RDF framework and Ontologies in general (there is nothing in this argumentation specific to science), although projects such as CSML point to conceptual spaces being particularly emphasized as semantic models for *scientific* data. We too have focused in this direction, though we would contend that a rigorous semantics for scientific data could potentially be generalized to shared/networked information in many domains, akin to a Semantic Web. In this concluding section, though, we intend to consolidate our discussion by focusing again on scientific data models.

It could be argued that the process of sharing scientific data does not actually require a data *semantics*. Most scientific data is in a tabular, vector, or matrix form, which (taken out of context) is just a jumble of numbers; it is only when plugged in to the appropriate software, or evaluated in its intended theoretical context, that these numbers actually become part of "science." In this case, one might argue, there is no particular reason to express scientific contexts within data sets themselves; data sets should simply record the minimum information needed to send or reproduce data across different research environments.

Contrary to that assessment, however, there *have* been numerous efforts in the research/academic community to codify scientific data models (some of which we listed in Section 2.2). At least some scientists, in short, have argued that published scientific data should be organized and annotated in a manner which documents scientific assumptions and contexts, rather than just serializing raw numbers. To the degree that researchers seek to formulate *expressive* data sets along these lines, then issues of "scientific data semantics" come to the fore. Exploring semantic theories to support such a data-semantics can draw in considerations from both science and digital technology, and even from linguistics and philosophy.

With that said, there is only a limited amount of information that can be provided via "static" structures within which data is encoded. A lot of scientific information (like information in general) is naturally expressed as *records*, tuples of individual fields with their own names or labels (**PatientName** and so forth), so at the very least field-names provide a conceptual overview of data semantics—indeed, this is the primary source of data-integration architectures within broad-based biomedical projects such as OMOP or CDISC (see Chapter 2). Controlled vocabularies attempt to render semantics based (primarily) on field-names (or, analogously, on column-names in the case of tabular data) more rigorous by ensuring that multiple parties use the same names or labels for conceptually similar units of information. Semantic Web Ontologies also build off of field-names by stipulating common axioms constraining information culled from different sources which utilize restricted field-names derived from controlled vocabularies; in this case, not only the textual *name* of the field, but also certain structural contracts in the data associated with the field (for instance, that a given field, e.g., patients' *first* name, is always paired with other fields, e.g., *last* name) is aligned between disparate data-providers.

Relatively "static" data models can also define and distinguish one-to-one, one-to-many, many-to-one, and many-to-many relationships (viz., different forms of relation cardinality), which provides another source of general conceptual overviews of data structures. For example, a patient presumably has only one (full) name, but they may be taking multiple medications. Likewise, pharmaceuticals may have only

one chemical formula, but they may be taken by many patients (in a clinical study, for example).

Yet (as we argued above at the end of Section 9.3.1) one can achieve only a limited degree of semantic precision via "static" details about (or meta-models constraining) data structures, such as field-names or relation-cardinality. This appears to be the gist of conceptual space arguments against the Semantic Web, and as such motivates proposals for more detailed statistical annotations, addressing issues such as units of measurement, valid ranges, the fusion of *dimensions* into *domains*, and similar meta-modeling constructions. Still, these are essentially static forms of meta-data, even if they lend greater theoretical precision than just "raw numbers."

Over and above static meta-data, we contend that a well-motivated data semantics will be predominantly "procedural," by which we mean that the actual "semantics" of scientific data—the empirical, theoretically informed meanings or interpretations one assigns to numeric quantities or other information-values measured or observed as part of a scientific investigation—depends on computations, where those values are analyzed. Any static meta-data, from field-names to dimensional annotations, can provide only a summarial precis of research data's scientific meaning. The full-fledged scientific significance of a given data structure depends on the theory and investigations where it originates, and to the degree that such research has a digital residue, it would lie in the set of procedures (algorithms, calculations, and so forth) that project a scientific model into the computational domain. In this sense, it is unrealistic to propose a *semantics* for scientific data *except* in the context of procedure-collections (e.g., code libraries) that manipulate such data.

This hypothesis has consequences from both theoretical and practical perspectives. Practically speaking, one entailment that seems to follow—a conclusion we would certainly advocate—is that code reuse is an intrinsic part of data sharing. It is now common practice for scientists to deposit raw research data in a public archive, and while such transparency and data-availability are preferable to the alternative (where data is not published at all), scientists should be encouraged to develop their data sets as "Research Objects" or similar "FAIR" resources where data and code are bundled together. This adds a burden to the publishing process, which should not be underestimated: preparing a reusable code base could easily become the most time-consuming part of a research project. But scientists can turn to reusable code as a forum for demonstrating their theories and methods in an interactive, data-driven fashion. Moreover, we will argue that data-set implementations can facilitate scientific projects even prior to the stage of open data sharing.

9.5.1 Research data and data integration

In this book we have employed the summarial figure of a "Semiotic Saltire" to describe a typical pattern in the *organization* of procedures within a multi-faceted code base (e.g., integrated software components such as "multi-aspect modules," returning to terminology from Chapter 4). To the degree that code accompanying scientific data covers multiple software-engineering concerns, it is likely that the resulting procedures will end up grouped into aspects according to a pattern similar to the Saltire, which in this case can potentially serve as a rough guide to identifying coding requirements. (That is, we intend the saltire model to be partly normative—this is how modules often *should* be organized—but also partly observational, i.e., procedures *tend* to group into such a pattern.) When preparing the code base for research data, concerns identified in the saltire tend to be foregrounded: how should the published data appear in GUIs? How should it be serealized and deserialized? How should it be structured for database persistence?

Though we assume that such a "structuring" of requirements applies to shared research data,

we would render similar analyses for contexts such as clinical research networks or multi-site clinical trials. During a trial's planning stages, for example, investigators might benefit from modeling the information generated during the course of the trial as a *de facto* research data set (of course, when trial results then become published academically some of that data will in fact *be* released as research findings). Data-set semantics applies more generally than just in the context of research results curated as open-access data sets; conceiving clinical-trial data as a still-emerging *research* data set can help structure the programming and data-collection logistics governing how the trial will digitally operate, and how (assuming a multi-site project) information from different institutions will be aggregated.

Insofar as research and/or clinical data are curated according to the norms of featureful open-access data sets (e.g., Research Objects), and if the resulting code base accepts the basic premise of multi-aspect design, then the resulting information resources will have from the outset a programming environment equipped with a useful variety of software capabilities: custom GUI classes, implemented protocols for data (de)serializing/marshaling, and so forth. Individual clinical trials, for example, can be encapsulated in distinct *modules,* which could be injected into clinical applications. All of this structure might then be leveraged for scenarios such as machine-learning or data-mining/integration.

For example, consider the task of unifying results from two different multi-site clinical trials. If each trial's data comes packaged in self-contained modules spanning multiple programming aspects, developers would have a valuable body of code already implemented for each information-space, which is more convenient than needing to write code *de novo* when presented with research or trial results as raw data. As a concrete example, suppose one stage of data integration requires the use of GUI components for human users to provide feedback about how two distinct data sets should be merged. If GUI classes are provided as part of the original data sets' modules, they would not have to be engineered from scratch within a data-integration context.

Aside from such practical maxims, however, this book has also focused on data semantics from a natural-language perspective, particularly that of conceptual spaces. How can we draw intuitions from natural language in the context of a predominantly *procedural* semantics? If the interpretive substance behind any scientific data only emerges in the context of procedures where that data is manipulated and analyzed, it would seem difficult to press into service any "static" meta-model (whether based on conceptual spaces, on Web Ontologies, or anything else), which would be logically removed from procedures themselves. The only real "semantics" applicable to a given scientific data-space would need to be assessed by looking at the specific procedures implemented for that data and how they exemplify the relevant scientific model/theory, through algorithms and data constructors.

In short, our discussion leads to the problem of formulating a *procedural* conceptual space semantics, to the degree that we wish to sustain the intuition of conceptual space theory as a richer and more realistic semantics alternative to (say) the canonical Semantic Web. We will therefore conclude by sketching a few ideas in this context.

9.5.2 Toward a procedural conceptual-space semantics

The essential insight of "procedural" semantics, at least in the context of scientific data, is that meanings and interpretations of scientific measurements/observations are dependent on the specific scientific theories guiding research where the data originates, and these are only computationally manifested in any substantial

way through procedures (algorithms, analyses, visualizations, and so forth). There is not a lot of semantic detail that can be provided by overarching computing environments *apart from* procedures implemented for each specific domain. A programming environment may provide tools to *facilitate* implementations, but in the absence of procedures actually composed in concrete fashion (expressing theoretical axioms, calculations, or algorithms via source code) we cannot attribute a significant *semantics* to such programming environments. The question then becomes apparently: granted that the semantic weight of a data-model rests predominantly on procedures formulated for an associated code model, how can the programming environment and computational tools which enable those procedures to be implemented at least *reinforce* the semantic details encapsulated via procedures themselves?

To examine this question, we'll point out first of all that procedures tend to cluster into interrelated groups. Moreover, this clustering effect is often correlated with data structures insofar as they would be represented within a data model. Consider the general case of multi-valued data fields (i.e., one-to-many relationships), such as the list of a patient's medications. Multi-valued collections require several different procedures to be fully manipulated, at the minimum those for *inserting* and for *removing* values. So (reprising Section 6.3.2) code managing a patient's health records, for example, might include a procedure that has the effect of *adding* reference to a certain medicine to the list which the patient is currently taking, and a second procedure for *removing* a medication from that list. These two procedures, of course, are logically inter-related. This is a simple example of how meta-modeling paradigms often propagate to *inter-procedural* relations, a tendency we discussed in Chapter 6 in the context of using code models to *instantiate* data models.

Chapter 6 also discussed data-modeling techniques such as scale/dimension annotations or remote procedure calls (i.e., modeling the preparatory code needed to expose procedures, or capabilities dependent on specific sequences of procedures, as an external service; we proposed the term "meta-procedure"). These likewise furnish examples of how *networks* of procedures concretize data-model paradigms. For example, validating scientific scales and units of measurements may involve preparatory code that ensures dimensional alignment as a precursor to performing some specific calculation, leading to functionality being split between two contexts: a "preparatory" procedure which performs a function we might refer to as "gatekeeping" [13], and then the "primary" procedure which enacts the requisite computations. Here the logic governing the relation between the "gatekeeping" and "primary" procedures manifests data-modeling concerns (in this case those of dimensional analysis and consistency), analogous to the case of multi-valued data fields being supported by both *insertion* and *deletion* procedures.

Given these implicit logical connections between procedures, programmers have the option of *explicating* such connections via code annotations or other interface-description techniques. Insofar as procedure-interrelationships concretize and originate from data-modeling concerns, notating procedural connections likewise serves the goal of transparently describing data models operating in the context of the current code base. In short, tools for constructing and identifying procedural annotations, particularly insofar as these allow procedure's clustering patterns to be described and rationalized, serve as one technology for elucidating data semantics through code components/modules.

There are multiple criteria that could be applied when modeling how procedures within a given library or component are interconnected. One can trace dynamic *execution flows* in the sense of identifying, for a given run of a program/application, which procedures are called prior or subsequent to which others. We can

also consider (more generally) which procedures *might* be called prior to others. Of course, one straightforward inter-procedural relation is when one procedure calls a second. Alternatively, an enclosing procedure might call an antecedent and then subsequent procedure in sequence. These cases are distinguished in terms of whether the prior procedure returns before the later one begins/resumes. A sufficiently expressive pointcut expression language (as we explored in Section 6.3) can identify locations in source-code where specific kinds of inter-procedure relations are exercised (one calling a second, one being called before a second in an overarching procedure, and so forth).

In and of itself, such program-flow connections do not necessarily have a specific "semantic" interpretation—they may merely reflect operational sequences as specific algorithms or implementation patterns are encoded—but at least on some occasions there is a meaningful semantics behind inter-procedural connections manifest at the program-sequence level (an example would be "gatekeeping" checks as mentioned earlier; another would be deserializing input data structures as preparation for handling a remote meta-procedure invocation). Insofar as pointcut expressions can single out code sites which *do* have such semantically substantial rationales, the use of pointcuts to construct rigorous code models can provide a technique for clarifying how data semantics are manifest within a code base, establishing the interactions between data and code models that we discussed in Chapter 6.

We might envision code libraries/modules as "procedural" spaces, whose underlying structures are constituted by semantically meaningful inter-procedural relations that can be annotated and described. A *procedural* space is, of course, not the same thing as a "conceptual" space, but this chapter has reviewed how conceptual space semantics are often formalized by considering the mutation in information content as one moves between sites of "transformations" that may be seen as analogous to procedures—procedures themselves in the context of computer code, or "morphisms" in hypergraph categories, and verbs in natural languages. In other words, a conceptual space semantics often emerges from networked procedure-spaces, or representations structurally akin to them.

Our discussion in this chapter has attempted to highlight one potential avenue for deriving a rigorous conceptual space semantics in a procedure-network context (whether this is defined explicitly or implicitly) through the lens of information-content "amplification" and "delta" paths/roles. We suggest that this is a promising avenue for future research, even if our analysis to this point only presents the initial step to a theory along such lines. Probably the trajectory of such a theory's development cannot be driven by abstract concerns alone, but rather in a feedback circle informed by specific scientific data sets for which data-semantics can be assessed concretely, and through the implementation of code-annotation systems where inter-procedural connections can be notated and analyzed.

References

[1] Benjamin Adams, Martin Raubal, A metric conceptual space algebra, in: International Conference on Spatial Information Theory, 2009, pp. 51–68, https://pdfs.semanticscholar.org/521a/cbab9658df27acd9f40bba2b9445f75d681c.pdf.

[2] Jens Allwood, Semantics as meaning determination with semantic-epistemic operations, in: Jens Allwood, Peter Gärdenfors (Eds.), Cognitive Semantics: Meaning and Cognition, John Benjamins, 1999, pp. 12–28, https://benjamins.com/catalog/pbns.55.02all.

[3] Lucas Bechberger, Kai-Uwe Kühnberger, A comprehensive implementation of conceptual spaces, in: Artificial Intelligence and Cognition, Proceedings, 2017, http://ceur-ws.org/Vol-2090/paper4.pdf.

[4] Lucas Bechberger, Elektra Kypridemo, Mapping images to psychological similarity spaces using neural networks, in: Artificial Intelligence and Cognition, Proceedings, 2018, http://ceur-ws.org/Vol-2418/paper3.pdf.

[5] Ismaïl Biskri, Jean-Pierre Descles, Applicative and combinatory categorial grammar (from syntax to functional semantics), in: Ruslan Mitkov, Nicolas Nicolov (Eds.), Recent Advances in Natural Language Processing, John Benjamins, 1997, https://www.researchgate.net/publication/232754402_Applicative_and_Combinatory_Categorial_Grammar_from_syntax_to_functional_semantics.

[6] Elli Bleeker, et al., Agree to disagree: modelling co-existing scholarly perspectives on literary text, https://academic.oup.com/dsh/article-abstract/34/4/844/5576174?redirectedFrom=fulltext.

[7] Elli Bleeker, et al., Between flexibility and universality: combining TAGML and XML to enhance the modeling of cultural heritage text, http://ceur-ws.org/Vol-2723/short39.pdf.

[8] Greg Carlson, Thematic roles and the individuation of events, https://www.sas.rochester.edu/lin/people/faculty/carlson_greg/assets/pdf/them-roles-events.pdf.

[9] Stuart Chalk, SciData: a data model and ontology for semantic representation of scientific data, Journal of Cheminformatics 8 (2016), https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0168-9.

[10] Lucas Champollion, Covert distributivity in algebraic event semantics, Semantics & Pragmatics 9 (2016) 1–65, https://semprag.org/article/view/sp.9.15.

[11] Lucas Champollion, Overt distributivity in algebraic event semantics, Semantics & Pragmatics 9 (2016) 1–65, https://semprag.org/article/view/sp.9.16.

[12] Lucas Champollion, Parts of a Whole: Distributivity as a Bridge Between Aspect and Measurement, Dissertation, University of Pennsylvania, 2010, https://repository.upenn.edu/cgi/viewcontent.cgi?article=2117&context=edissertations.

[13] Nathaniel Christen, Hypergraph type theory for specifications-conformant code and generalized lambda calculus, in: Amy Neustein (Ed.), Advances in Ubiquitous Computing: Cyber-Physical Systems, Smart Cities, and Ecological Monitoring, Elsevier, 2019.

[14] Bob Coecke, et al., Interacting conceptual spaces I: grammatical composition of concepts, Extended version of Proceedings of the 2016 Workshop on Semantic Spaces at the Intersection of NLP, Physics and Cognitive Science, pp. 11–19, https://arxiv.org/pdf/1703.08314.pdf.

[15] Bridget Copley, Force dynamics, in: Robert Truswell (Ed.), Oxford Handbook of Event Structure, Oxford, 2019, pp. 103–149, http://bcopley.com/pubs/force-dynamics.

[16] Ann Copestake, et al., An architecture for language processing for scientific texts, in: UK e-Science Programme All Hands Meeting, Proceedings, 2006, https://abdn.pure.elsevier.com/en/publications/an-architecture-for-language-processing-for-scientific-texts.

[17] Suelen M. de Paula, Ricardo R. Gudwin, Evolving conceptual spaces for symbol grounding in language games, Biologically Inspired Cognitive Architectures 14 (2015) 73–85, https://www.sciencedirect.com/science/article/pii/S2212683X15000493.

[18] Ronald Haentjens Dekker, et al., Parsing a markup language that supports overlap and discontinuity, in: Document Engineering, Proceedings, 2020, pp. 1–4, https://dl.acm.org/doi/abs/10.1145/3395027.3419590.

[19] Stefan Dietze, John Domingue, Exploiting conceptual spaces for ontology integration, in: 3rd Asian Semantic Web Conference, 2008, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.4434&rep=rep1&type=pdf.

[20] Gilles Fauconnier, Mental Spaces: Aspects of Meaning Construction in Natural Language, Cambridge University Press, Cambridge, 1994.

[21] Gilles Fauconnier, Mark Turner, The Way We Think: Conceptual Blending and the Mind's Hidden Complexities, Basic Books, 2002.

[22] Johannes C. Flieger, Gradable Adjectives and the Semantics of Locatives, Dissertation, University of Edinburgh, 2009, https://era.ed.ac.uk/bitstream/handle/1842/3995/Flieger2009.pdf?sequence=1&isAllowed=y.

[23] Peter Gärdenfors, Does semantics need reality?, in: Alexander Riegler, et al. (Eds.), Understanding Representation in the Cognitive Sciences, Springer, Boston, 1999, pp. 209–217.

[24] Peter Gärdenfors, How to make the semantic web more semantic, https://slab.org/tmp/Gardenfors04.pdf.

[25] Peter Gärdenfors, Primary cognitive categories are determined by their invariances, Frontiers in Psychology (2020), https://www.frontiersin.org/articles/10.3389/fpsyg.2020.584017/full.

[26] Peter Gärdenfors, Massimo Warglien, Using conceptual spaces to model actions and events, Journal of Semantics 29 (4) (2012) 487–519, https://www.researchgate.net/publication/274999478_Using_Conceptual_Spaces_to_Model_Actions_and_Events.

[27] Gabriel Gaudreault, Derivational Event Semantics for Pregroup Grammars, Dissertation, Concordia University, Montreal, 2016, https://spectrum.library.concordia.ca/981873/9/Gaudreault_MA_F2016.pdf.

[28] T. Florian Jaeger, Redundancy and reduction: speakers manage syntactic information density, Cognitive Psychology 61 (1) (2010) 23–62, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2896231/.

[29] Ronald Langacker, Cognitive Grammar: A Basic Introduction, Oxford University Press, Oxford, 2008.

[30] Ronald Langacker, Interactive cognition: toward a unified account of structure, processing, and discourse, International Journal of Cognitive Linguistics 3 (2) (2014), http://lchc.ucsd.edu/MCA/Mail/xmcamail.2014-08.dir/pdf8jEY9UCVVh.pdf.

[31] Ronald Langacker, The English present: temporal coincidence vs. epistemic immediacy, in: Adeline Patard, Frank Brisard (Eds.), Cognitive Approaches to Tense, Aspect, and Epistemic Modality, John Benjamins, 2011, https://benjamins.com/catalog/hcp.29.06lan.

[32] Beth Levin, Malka Rappaport Hovav, Lexicalized meaning and manner/result complementarity, in: B. Arsenijević, et al. (Eds.), Studies in the Composition and Decomposition of Event Predicates, Springer, 2013, pp. 49–70, https://web.stanford.edu/~bclevin/barcel11rev.pdf.

[33] Ian Lewin, Using hand-crafted rules and machine learning to infer SciXML document structure, https://www.semanticscholar.org/paper/Using-hand-crafted-rules-and-machine-learning-to-Lewin/3dd2756e42ae24df0769711c3b7a55249d7b17cf.

[34] Peter McQuilton, et al., BioSharing: curated and crowd-sourced metadata standards, databases and data policies in the life sciences, Database 2016 (2016), https://pdfs.semanticscholar.org/504a/e934c99ed5cbd4ea925ec13ebf86f553e251.pdf.

[35] Carlos Montemayor, et al., Implementation, formalization and representation: challenges for the integrated information theory, Journal of Consciousness Studies 26 (1–2) (2019) 107–132, http://online.sfsu.edu/barros/publications/publications/files/MontemayorEtAl2019.pdf.

[36] Erwan Moreau, From link grammars to categorial grammars, in: Proceedings of Categorial Grammars 2004, 2004, pp. 31–45, https://hal.archives-ouvertes.fr/hal-00487053/document.

[37] Tiago Nunes, et al., BeCAS: biomedical concept recognition services and visualization, https://www.ncbi.nlm.nih.gov/pubmed/23736528.

[38] Matías Osta Vélez, Inference and the Structure of Concepts, Dissertation, Ludwig-Maximilians-University, 2020, https://edoc.ub.uni-muenchen.de/27633/7/Osta_Velez_Matias.pdf.

[39] Joey Pollack, Holism, conceptual role, and conceptual similarity, https://www.tandfonline.com/doi/abs/10.1080/09515089.2020.1729973?journalCode=cphp20.

[40] Dietrich Rebholz-Schuhman, et al., IeXML: towards an annotation framework for biomedical semantic types enabling interoperability of text processing modules, https://www.semanticscholar.org/paper/IeXML%3A-towards-an-annotation-framework-for-semantic-Rebholz-Schuhmann-Kirsch/1d72a56b6576117c62f388a5f2193965e4c7e293.

[41] John T. Rickard, A concept geometry for conceptual spaces, Fuzzy Optimization and Decision Making 5 (2006) 311–329, https://altexploit.files.wordpress.com/2017/06/a-concept-geometry-for-conceptual-spaces.pdf.

[42] Bradley Rives, The empirical case against analyticity: two options for concept pragmatists, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.188.9556&rep=rep1&type=pdf.

[43] Aurélie Rossi, Applicative and combinatory categorial grammar: analysis of the French interrogative sentences, in: FLAIRS Conference, Association for the Advancement of Artificial Intelligence, 2008, https://www.aaai.org/Papers/FLAIRS/2008/FLAIRS08-118.pdf.

[44] C.J. Rupp, et al., Flexible interfaces in the application of language technology to an eScience corpus, https://www.cl.cam.ac.uk/~sht25/papers/Rupp_et_al.pdf.

[45] Dan Ryder, Problems of representation II: naturalizing content, in: The Routledge Companion to Philosophy of Psychology, Routledge, 2009, pp. 251–279, https://www.semanticscholar.org/paper/Problems-of-representation-II%3A-naturalizing-content-Ryder/1073bf288423e3e9b1940628cb2deb7db8b8fae2.

[46] Bahar Sateli, René Witte, Semantic representation of scientific literature: bringing claims, contributions and named entities onto the linked open data cloud, PeerJ Computer Science (2015), https://peerj.com/articles/cs-37.pdf.

[47] Pedro Sernadela, José Luís Oliveira, A semantic-based workflow for biomedical literature annotation, Database 2017 (2017), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5691355/.

[48] Frank Andreas Sposito, What do data curators care about? Data quality, user trust, and the data reuse plan, http://library.ifla.org/1797/1/S06-2017-sposito-en.pdf.

[49] Gregor Strle, Semantics Within: the Representation of Meaning Through Conceptual Spaces, Dissertation, Novi Gorici, 2012, http://www.ung.si/~library/doktorati/interkulturni/25Strle.pdf.

[50] Anna Maria Tammaro, et al., Data curator's roles and responsibilities: an international perspective, Libri 69 (2) (2019), https://www.degruyter.com/document/doi/10.1515/libri-2018-0090/html.

[51] Miriam Taverniers, Subjecthood and the notion of instantiation, https://semanticsarchive.net/Archive/DU0OGRkO/Taverniers-2005-Subjecthood-PREPRINT.pdf.

[52] Chris F. Taylor, et al., Promoting coherent minimum reporting guidelines for biological and biomedical investigations: the MIBBI project, Nature Biotechnology 26 (2008) 889–896, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2771753/.

[53] Simone Teufel, Min-Yen Kan, Robust argumentative zoning for sensemaking in scholarly documents, in: R. Bernardi, et al. (Eds.), NLP4DL/AT4DL, in: Advanced Language Technologies for Digital Libraries, 2009, pp. 154–170, http://people.cs.pitt.edu/~litman/courses/nus062615/readings/TeufelKan.pdf.

[54] Simone Teufel, et al., An annotation scheme for discourse-level argumentation in research articles, in: Ninth Conference of the European Chapter of the Association for Computational Linguistics, 1999, pp. 110–117, https://www.aclweb.org/anthology/E99-1015/.

[55] Ralf Vogel, Polyvalent Verbs, Dissertation, Humbolt University, 1998, https://edoc.hu-berlin.de/bitstream/handle/18452/15160/Vogel.pdf?sequence=1.

[56] Massimo Warglien, Peter Gärdenfors, Matthijs Westera, Event structure, conceptual spaces and the semantics of verbs, Theoretical Linguistics 37 (2012) 159–193, https://iris.unive.it/retrieve/handle/10278/37082/27662/semantics_of_verbs.pdf.

This page intentionally left blank

Index

D

S

T

U

V

W

X

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care

Amy Neustein and Nathaniel Christen

In recent years, scientific research and translation medicine have placed increased emphasis on computational methodology and data curation across many disciplines, both to advance underlying science and to instantiate precision-medicine protocols in the lab and in clinical practice. The nexus of concerns related to oncology, cardiology, and virology (SARS-CoV-2) presents a fortuitous context within which to examine the theory and practice of biomedical data curation.

Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care argues that a well-rounded approach to data modeling should optimally embrace multiple perspectives inasmuch as data-modeling is neither a purely formal nor a purely conceptual discipline, but rather a hybrid of both. On the one hand, data models are designed for use by computer software components, and are, consequently, constrained by the mechanistic demands of software environments; data modeling strategies must accept the formal rigors imposed by unambiguous data-sharing and query-evaluation logic. In particular, data models are not well-suited for software-level deployment if such models do not translate seamlessly to clear strategies for querying data and ensuring data integrity as information is moved across multiple points. On the other hand, data modeling is, likewise, constrained by human conceptual tendencies, because the information which is managed by databases and data networks is ultimately intended to be visualized/utilized by humans as the end-user.

Thus, at the intersection of both formal and humanistic methodology, data modeling takes on elements of both logico-mathematical frameworks (e.g., type systems and graph theory) and conceptual/philosophical paradigms (e.g., linguistics and cognitive science). The authors embrace this two-sided aspect of data models by seeking non-reductionistic points of convergence between formal and humanistic/conceptual viewpoints, and by leveraging biomedical contexts (viz., COVID, Cancer, and Cardiac Care) so as to provide motivating examples and case-studies in this volume.

Key Features:

- Provides an analysis of how conceptual spaces and related cognitive linguistic approaches can inspire programming and query-processing models
- Outlines the vital role that data modeling/curation has played in significant medical breakthroughs
- Presents readers with an overview of how information-management approaches intersect with precision medicine, providing case studies of data-modeling in concrete scientific practice
- Explores applications of image analysis and computer vision in the context of precision medicine
- Examines the role of technology in scientific publishing, replication studies, and dataset curation

About the Authors:

Amy Neustein, CEO and Founder of Linguistic Technology Systems in Fort Lee, NJ, USA, a think tank for database engineering, scientific computing, and programming language theory. She is the Volume Editor of *Advances in Ubiquitous Computing: Cyber-Physical Systems, Smart Cities, and Ecological Monitoring* published by Elsevier in 2020.

Nathaniel Christen, Lead Software Architect at Linguistic Technology Systems in Fort Lee, NJ, USA. He is a doctoral candidate (on leave) at the University of Ottawa in Canada. His doctoral research and dissertation have focused on the interrelation of phenomenology, cognitive linguistics, and the philosophy of science.

ELSEVIER

ACADEMIC PRESS

An imprint of Elsevier

elsevier.com/books-and-journals

ISBN 978-0-323-85197-8