Foundations and Trends® in Information Retrieval

[DRAFT]

Mathematical Information Retrieval

Searching with Formulas and Text


Richard Zanibbi
Department of Computer Science
Rochester Institute of Technology
rxzvcs@rit.edu

Behrooz Mansouri
Department of Computer Science
University of Southern Maine
behrooz.mansouri@maine.edu

Anurag Agarwal
School of Mathematics & Statistics
Rochester Institute of Technology
axasma@rit.edu

This article may be used only for the purpose of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval.
# Contents

## Preface

### 1 Sources, Tasks, and Search

1.1 When and where do we search? .................................. 5
1.2 Information task framework ........................................ 7
1.3 Information needs and search strategies ......................... 12
1.4 Retrieval systems .................................................. 20
1.5 User-system interactions and interfaces ......................... 25
1.6 Summary ............................................................ 34

### 2 Representing Formulas and Text

2.1 What information does a formula carry? ......................... 36
2.2 Formula representations ............................................ 45
2.3 Formula annotation and indexing .................................. 56
2.4 Indexing formulas and text ........................................ 66
2.5 Summary ............................................................ 70

### 3 Retrieval Problems and their Evaluation

3.1 How do we evaluate math IR systems? .......................... 71
3.2 Evaluation Metrics for Search Results ............................ 82
3.3 Creating Test Collections ......................................... 83
3.4 Formula Search: Tasks and Evaluation .......................... 85
ABSTRACT

Mathematical information is essential for technical work, but its creation, interpretation, and search are challenging. To help address these challenges, researchers have developed multimodal retrieval models and interfaces supporting both formulas and text in search. This book begins with a simple framework characterizing the information tasks that people and systems perform as we work to answer math-related questions. The framework is used to organize and relate the other core topics of the book, including interactions between people and systems in math-aware search, representing math formulas in sources, and retrieval models for formulas and for formulas and text. Additional topics include math question answering and evaluation techniques. We close with some key questions and concrete directions for future work. This book is intended for use by students, instructors, and researchers, and those who simply wish that it was easier to find and use mathematical information.
This is an early draft of the book. The first two chapters are complete, and we have included the table of contents for additional chapters that we are currently working on to provide a sense of the final organization of the book.

The first full draft of the book will be complete by the end of summer 2024. If you have any comments questions, please feel free to contact the authors at rxzvcs@rit.edu.

Best wishes,

Richard Zanibbi
Behrooz Mansouri
Anurag Agarwal
We often pause to search when something that we read, watch, or hear prompts questions that we want answers to. We then go about finding answers using additional sources of information: some already exist, some are created in response to requests (e.g., emails or search results), and some are created to record and organize what we find along with partial or complete answers to our questions.

In this way, information sources are the backbone of our information ecosystems. The sources available in our information ecosystem place a hard limit upon which questions we can answer. In addition to the information content in a source, its terminology, notation, writing style, and other factors determine the amount of information one can recover from a source, and how accurately and completely. This is a key reason why math instructors that communicate well are so highly regarded: they help us more easily understand topics by how they speak, write, and present exercises. Through course materials, lectures, and conversations, these instructors provide multiple sources tailored to their students’ level of understanding and communication style.

Outside the classroom, we still often find ourselves in need of mathematical information. It might be as simple as finding a formula to
convert temperatures in Fahrenheit to Celsius, or the formula associated with a name (e.g., inverse document frequency). Or the goal may be more complex, such as understanding a proof of the sensitivity conjecture.

As we look for answers, in addition to collecting sources that we find, we will in some way annotate and organize them to identify and apply pertinent information, e.g., to find other sources, choose different search terms, execute suggested exercises, and make notes about partial answers to our questions. The effort needed for these tasks depends largely on the content and presentation in the sources that we access, including search results and discussion online and in-person with helpful people. To save time, we will often annotate sources and create additional sources of our own (e.g., bookmarking a web page, placing notes in a file, or highlighting a PDF document).

In this book, when we speak about sources, we are referring to individual documents, recordings (e.g., videos) or other artifacts that contain information directly. Libraries and other people are of course also information sources, in the sense that they can provide information, but here we use 'sources' to refer to records of specific information.

To help organize our study of mathematical information retrieval, in this chapter we introduce a framework for information tasks based on sources, and the different ways that we retrieve, analyze, and use sources to answer mathematical questions. This information task framework is built upon two main ideas:

1. Search begins, progresses, and ends with sources.
2. Tasks other than search are often needed to find information.

The key components of the framework are:

- **information needs** that individuals have,
- **sources of information** that we search, consult, and create,
- information **tasks** performed to address information needs, and
- their roles in search algorithms and user-interfaces.

In the next section, we consider how these components interact when
we have a mathematical question that we wish to answer.

1.1 When and where do we search?

Some short answers to this question are (1) when we have a question, and (2) wherever is easiest. While not very satisfying, these answers are basically correct. Search is generally performed as part of some larger information task, and not for its own sake.\(^1\) This motivates finding quick paths to answers.

However, technical subjects such as mathematics can be complex. Finding and understanding information on math may require multiple activities, such as web search, reading sources (e.g., Wikipedia pages and textbooks), taking notes, talking to instructors or colleagues, and doing exercises. As a result, when retrieving technical material on math and other specialized topics (e.g., law, chemistry, music history), it is helpful to understand how search interacts with other information tasks.

To illustrate, consider the more general problem of sensemaking, which learning about detailed mathematical topics is closely related to.\(^2\) In sensemaking, we construct a conceptual understanding of a topic with many sources, usually along with communicating this understanding. Common examples include writing a school term papers on an unfamiliar topic (e.g., applications of category theory), or summarizing a complex historical event from multiple news reports.

Sensemaking tasks are challenging because information must be found in multiple sources, but also because this information must be analyzed, compared, and integrated. These thinking activities often require most of the effort for sensemaking. To manage these thinking tasks, we record plans, notes, and outlines to organize our work. These working documents may be checked repeatedly as we work, and as we write our final summary. They are themselves important information sources that provide the scaffolding needed to focus and ultimately complete work on a sensemaking task.

To further illustrate information tasks that complement search, imagine taking handwritten notes on eigenvectors as described in an

---

\(^1\)A fact that is both important and humbling for IR researchers.

\(^2\)See (Hearst, 2009) for an overview of early research on sensemaking.
algebra textbook. The notes allow us to annotate this source with our own observations, and record them for reference at a later time. The analysis and insights in the notes come from applying information that we know and find. These notes communicate a new information source to a specialized audience: ourselves.

For our notes to be useful, we organize them. Perhaps this is a purple sticky note that we attach to a monitor to consult later in the evening, after dinner. Or, perhaps we use a paper notebook with separate sections for different subjects, along with other organizational devices (e.g., sticky notes acting as bookmarks). We might instead be using a tablet computer, which also provides handwriting recognition to convert the notes to computer-searchable data (e.g., using Ctrl-f).

The information tasks above are distinct from a basic search task where we submit a query, post a question, or send an email to obtain new information sources. However, it turns out that search engines implement variations of the same basic tasks described above: they need to index, communicate, annotate, and apply information in sources to be effective. For example, we organize sources when we arrange sticky notes by topic and color on a wall, or construct an inverted index mapping words or formulas to their document locations: these are both forms of indexing. As another example, search engines produce Search Engine Result Pages (SERPs) summarizing documents matching a query, and question answering systems or AI ‘bots’ produce answers. These retrieval system outputs and our notes are communications creating new information sources.

Making notes on a passage requires us to apply information to create an annotation: additional information associated with the passage. In turn, if those notes were handwritten on a tablet computer, a system converting these to text and LaTeX for math applies information captured in an algorithm, annotating the notes themselves. We end up with a hierarchy of annotations: the notes annotate a passage, while a recognition algorithm annotates the notes.

In our framework we will distinguish different source types, based largely on what information tasks they are primarily used for. More specifically, we distinguish:
1. Information task framework

1. available sources on a topic including search queries and results,
2. information added to sources (annotation), and
3. structures and organizations created for search (indexing)

Getting back to our motivating question, when we have identified a mathematical information need, we generally start with questions, and hope to end with one or more information sources that we feel address or ideally answer those questions (i.e., relevant sources for the information need). Where we search is motivated by the types of sources we expect to find from places online and/or the physical world (e.g., conversations and post-its). Unless we are casually browsing resources on a topic, the places and order in which these sources are found will generally reflect attempts to reduce our time and effort. Relevant sources are often of different types: perhaps a passage in a web page along with a SERP page, an answer from an online AI system, an email from a friend, and a green sticky note on your monitor.

From this perspective, math-aware search engines and question answering systems are important tools, but only one among many resources for finding math information, and only a small part of what happens when we search.

1.2 Information task framework

While we focus in this book on information retrieval using computers, we wish to address sources in their broadest sense here. Not all sources are text documents, and not all sources are recorded in documents. Consider an informal conversation about Bayesian decision theory in the hallway, or observing that there are no clouds in the sky: often, your only record of important information sources is your own memory.

In addition to textbooks, technical papers, and web pages, in recent years the types of resources used to locate mathematical information has grown to include substantial amounts of video (Davila et al., 2021) and audio, e.g., for course lectures, tutorials, and technical talks. Community Question Answering sites, and direct question answering is provided by

---

Information foraging theory (Pirilli and Card, 1999) suggests we evolved to gather and consume information similar to food, governed by cost-benefit analyses.
resources such as Math Stack Exchange,\textsuperscript{4} Wolfram Alpha, and large
language models such as the Generative Pre-trained Transformer (GPT).

It is worth noting that when we have found or produced information
we want to share or reuse, we usually produce a source of information
ourselves. For example, when we have an answer to a math homework
question, we create a physical or digital document, so that this can be
checked by ourselves and graded by our instructor. If we found a helpful
video while doing the homework, we might share it in a text message,
which is itself a form of ‘micro-source.’

Especially when we include information obtained directly from our
environment along with modern computing and communication devices,
information sources may come in many forms. Sources vary in the
dimensions listed below, among others.

\textbf{Differences in Information Sources}
\begin{itemize}
\item immediacy, \textit{e.g.}, having a conversation vs. reading a transcription
\item authorship, \textit{e.g.}, human, machine generated, environment
\item interactivity, \textit{e.g.}, a human/chabot conversation vs. a document
\item audience, \textit{e.g.}, grade school students vs. math professors
\item modality, \textit{e.g.}, text, video, audio, or a web page combining these
\item purpose, \textit{e.g.}, textbook, search results, or a search index
\item structure, \textit{e.g.}, free text in a sticky note, vs. a book with chapters
\item length
\item formality, \textit{e.g.}, proof vs. text message
\item style, \textit{e.g.}, how concepts and examples are communicated
\item correctness, \textit{e.g.}, correct vs. incorrect definition or proof
\item completeness, \textit{e.g.}, partial vs. full search index
\end{itemize}

For the \textit{immediacy} of a source, we are referring to whether the source
comes directly from observing a person’s environment (\textit{e.g.}, through con-
versation, experimentation, travel, etc.) or is recorded, as in a document
or audiovisual recording. Particularly with the advent of large language
models, authorship and correctness are important concerns. In many
cases LLMs and other sources may appear credible but are incorrect.

\textsuperscript{4}https://math.stackexchange.com
Knowing how a source is created can help us determine how trusting we should be of it when we are uncertain about validity (e.g., from the perceived expertise of an author or system).

The intended audience and style of a source are also critical concerns. They determine the prerequisite knowledge needed to decide whether a source is relevant and to interpret and use information in the source.

**Information task types.** The primary task types we will use for people and systems come from common descriptions of sensemaking and simpler information tasks: retrieving, analyzing, and synthesizing. For this framework, these tasks are considered at the source level. For example, analyzing a source refers to analysis that produces additional recorded information for a source (e.g., in a note) rather than reading and interpreting a source, which does not produce a new observable artifact.

We subdivide each of these into two subtasks based on how sources are created and used, producing six tasks in total.

1. **Retrieve**
   - *Query* to request sources of information
   - *Consult* and interpret available sources, *examining* and *navigating* within and across sources

2. **Analyze**
   - *Annotate* sources with additional information, e.g., notes, add formula locations
   - *Index* sources by organizing them for retrieval

3. **Synthesize**
   - *Apply* available information that we know, have in available sources, or is encoded in algorithms, etc.
   - *Communicate* information in new sources

   The apply task is critical, and used for all other tasks (e.g., creating queries, consulting or creating information sources, or generating annotations and/or indexes). It is distinct because people often apply information without producing observable sources. For example, recognizing what a variable represents does not involve creating an an-
notation, index, or source outside of our own minds. The *apply* task also identifies an important commonality between thought and computation (*e.g.*, algorithms): both apply information but using differing levels of formality, flexibility, and automation.

Not all information needs require queries. If we have a helpful document describing the inverse document frequency on our laptop, we may simply consult it to review previously highlighted passages. This locating of items in an available source or across available sources through references and links is known as *navigation*, which is distinct from submitting a *query* to a system or person to find new sources. As another example of using navigation to satisfy an information need, in some case we may simply use the contents of available sources directly (*e.g.*, copying-and-pasting into an online form).

The process of analyzing a source and recording a map for use in retrieval is known as *indexing*. Consider Figure 1.1, where a book index provides a map for the book, so that a reader can quickly *navigate* to parts of the book discussing ‘Terms,’ for example. Contrast this *subject index* with the index used in a traditional term-based search engine, which provide a much simpler map known as a *concordance* recording where specific terms appear in documents (Duncan, 2021). While these
different indices are both used for retrieval, they differ in their scales (one document vs. a collection) and intended audiences (human reader vs. search algorithm). Other forms of indexing are less formal, such as collecting and organizing notes on different sticky notes for easier use.

As discussed earlier, we distinguish tasks for analyzing sources in terms of organizing them for use and retrieval (indexing), and adding information to sources (annotation). Annotations are often used in indexing sources, such as adding formula locations for PDF documents.

**Source Jar Framework.** To put sources and the tasks used to create them in a more intuitive relationship, Figure 1.2 visualizes our task framework as a jar of sources with a lid. The jar contains immediately available sources as marbles in the jar. Each marble has an identifying color and shape. The source marbles contain information of different types, and may refer to other sources inside and outside of the jar. Sources that are directly available are either with us, or inside the jar.

Stickers on the jar identify the information task types we can perform, and the lid is labeled with the need it is used to address. When we find or create a new information source, we add a marble to the jar. If a
new source *annotates* another source, we place it in a container with the source it describes inside the jar (e.g., using a small plastic box). *Indexing* produces a marble containing a description of which sources it organizes, and how. We take source marbles and containers out of the jar to use them. When they are no longer useful, we return them to the jar. It is also possible to lose sources when the jar is accidentally left open and ‘spilled.’

We can imagine having a shelf of these jars for different information needs. To reuse or get additional information for a need we worked on previously, we open a jar from our shelf. For a new information need we create a jar, adding any potentially useful sources from our other jars. When we stop working, we may select any final sources for use later, and then close the lid.

This informal jar model is intended to roughly capture how people experience working with information in a simple way. It captures observable sources and observable task actions. We tend to move from source to source, performing tasks of some specific types with a goal in mind. We are often unaware of why we performed tasks in a particular order, and so this is not represented explicitly, other than as marbles moving in and out of the jar through time.

### 1.3 Information needs and search strategies

**Information Needs** When searching for information on math, what we need to find will vary from finding definitions for terminology, math symbols, formulas, operational knowledge such as proof techniques, applications of mathematics (e.g., information retrieval models), resources for instructors, and detailed information on mathematical spaces, theorems, etc.

Example 1 illustrates information needs that different audiences may be seeking to address using the same query, along with a list of sources that might be used to address their needs. These needs vary from finding definitions to exploring sophisticated relationships between

---

5 *e.g.*, ‘the dog ate it,’ ‘my internet is down,’ or ‘I know it’s here...somewhere.’
6 *e.g.*, ‘Wait; I forgot one of the types of category theory applications I wanted to discuss in my paper from my notes...’
Example 1: Differing Information Needs

**Query:** What does \( a^2 + b^2 = c^2 \) represent and how is it useful?

*Students* might use this query to learn the Pythagoras theorem, and perhaps find an example demonstrating the theorem, and a possible proof.

*Educators* may have similar interests to students, but may seek additional resources on how to teach this result.

*Researchers* can have very different interests than the other audiences. They may be interested in one of more of the following:

- For a mathematician: Is this true in a general metric space and/or a Hilbert space?
- For a physicist: How is it linked to the probability assignments in quantum mechanics?
- For an IR expert: How is it related to probability assignments in a Hilbert space used in describing interaction for information retrieval?

**Possible Relevant Sources:** web-pages (MSE, Brilliant.org etc.), YouTube videos, online lecture slides, text documents (*e.g.*, digital books (OpenStax, LibreTexts), articles, online notes (MIT OpenCourseWare)).
the generalization of the theorem in different mathematical structures (Hilbert spaces) and applications in other fields (quantum mechanics).

As a result, the types of sources needed by each audience differ dramatically, but the initial (admittedly vague) query is identical: the query intent differs for these audiences. For math information needs, we have found it important to consider information needs both in terms of the desired information, as well as who is searching. Also shown in the example are relevant sources that might be used to address these needs from the different audiences, whether wholly or in part.

For a broader sense of the types of mathematical information needs users have online, Table 1.1 illustrates information needs for math organized by Broder’s taxonomy of needs/intents behind web search queries (Broder, 2002). While some question the usefulness of the transactional class in Broder’s model, for math, the transactional class is a useful distinction. For example, a user may be looking to \textit{refind} a web page they used to enter formulas in \LaTeX{} (i.e., a navigational intent). Or, they may instead be looking to find such a web page for the first time, thereby looking to interact/transact with as-yet unknown web sites (i.e., a transactional intent).

Within the \textit{informational} needs class, a distinct subclass of computational/operational information needs exist. These include needs to evaluate or simplify a formula, or to produce a proof for a statement using logical operations. It was useful to distinguish questions that were seeking \textit{concepts, proofs,} and \textit{computation} for the ARQMath shared tasks (Mansouri \textit{et al.}, 2022b) that we discuss in Chapter 3.

In our work we have found it useful to consider math information tasks in two dimensions, based on the type of information need as shown in Table 1.1, and the user’s mathematical background. More formally, we have a space/set of mathematical information needs $N$ defined by a Cartesian product of possible information needs ($T$)ypes and user/audience ($B$)ackgrounds ($N \in T \times B$). How these types and backgrounds interact is illustrated in Example 1.

\textbf{Search strategies.} For a given information need, it helps to think about \textit{strategies} that might be used to satisfy it. We can sketch these in strategy ‘jar’ diagrams as seen in the panel labeled Strategy 1. The
1.3. *Information needs and search strategies*

Table 1.1: Examples of Mathematical Information Needs within Broder’s Taxonomy (Broder, 2002)). A user’s math background is another dimension.

<table>
<thead>
<tr>
<th>Navigational</th>
<th>Find a specific source (‘known item’ retrieval)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Web page (e.g., for formula entry)</td>
</tr>
<tr>
<td></td>
<td>Document (e.g., Book, Technical Paper)</td>
</tr>
<tr>
<td></td>
<td>Video</td>
</tr>
<tr>
<td></td>
<td>Audio recording</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transactional</th>
<th>Find online resources for use/interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Formula entry</td>
</tr>
<tr>
<td></td>
<td>Evaluating and plotting a formula</td>
</tr>
<tr>
<td></td>
<td>Simplification of a formula</td>
</tr>
<tr>
<td></td>
<td>Interactive theorem proving</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Informational</th>
<th>Find information for a topic or question</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sub-categories: computation, concepts, and proofs</td>
</tr>
<tr>
<td></td>
<td>How to compute an expression (e.g., integral)</td>
</tr>
<tr>
<td></td>
<td>Symbol and operation definitions (e.g., $\zeta$, $\binom{n}{k}$)</td>
</tr>
<tr>
<td></td>
<td>Concept name(s) associated with a formula</td>
</tr>
<tr>
<td></td>
<td>When is a function not differentiable?</td>
</tr>
<tr>
<td></td>
<td>Who was Gauss?</td>
</tr>
<tr>
<td></td>
<td>Proof drafts for P = NP</td>
</tr>
</tbody>
</table>

diagrams identify an information need, initial queries, expected tasks, and where relevant sources might be found. We can imagine beginning a new search by writing the information need on the lid, putting already available source ‘marbles’ in the jar, and then writing planned tasks on the jar labels. For readability, we use informal descriptions for the three main task types along with initial queries. For this discussion, in our examples strategies we identify multiple relevant sources already known to the authors, some or none of which a searcher might actually know when they begin searching. An actual list of possibly relevant sources may be much shorter, or empty.

Let us first consider search strategies that might be used by undergraduate students, for learning how to complete a square, and to change the base of a logarithm (Strategy 1 and Strategy 2). In both examples, two queries that might be used are given, and the *Synthesis* tasks clarify
Sources, Tasks, and Search

Strategy 1: (Student) Completing the Square

Retrieve:
Query: completing the square OR \( ax^2 + bx + c = (\ast)^2 + \text{constant?} \)
Search using the text query or possibly the symbolic query; \( (\ast) \) is a wildcard for any subexpression. Identify where the general method can be found, and examine the proof of the result.

Analyze: Mark-up/bookmark sources to identify useful information. Use a notebook to summarize key details found in sources. Save examples for different cases, \( e.g., a, b > 0 \) and \( c \leq 0 \).

Synthesize: Solve an integration problem on paper, such as \( \int \frac{1}{x^2 + 4x + 3} \, dx \).

Possible Relevant Sources: web-pages (Math Stack Exchange (MSE), Brilliant.org etc.), YouTube videos, online lecture slides, text documents (\( e.g., \) digital books (OpenStax, LibreTexts), articles, online notes (MIT OpenCourseWare)), Online tutorial sites (Khan Academy, Magoosh), online answer engines (WolframAlpha), online calculators (SymboLab).

the specific information need: the source they want to produce. In the first example, this involves completing an exercise on paper, and in the second example, obtaining a value from a calculator. Note that for the queries containing formulas, students might find it difficult or be unable to express the formulas in queries using a standard text query box, particularly if they are unfamiliar with \LaTeX{}.

Now let’s consider more advanced information needs for researchers. The researchers may be interested in following progress on an old conjecture (\( e.g., \) Riemann Hypothesis). Or, they may be interested in learning about a new possible proof of the problem, or perhaps they were unfamiliar with the problem but are curious to know more about it. Strategy 3 seeks information and a proof for a problem that was posed in 1994. It became a major unresolved question in mathematical computer science until 2019, when Hao Huang solved it.

As another example, imagine that a researcher encounters a technical statement for the sensitivity conjecture, but which does not name it. They want to know the status of the statement, and if there are associated results they can use in their own work. Here the searcher only wants to learn the conjecture’s name, properties, and proofs for later reference. The strategy from Strategy 3 needs to be altered, as reflected in Strategy 4. In this second case, the researcher has a document
1.3. Information needs and search strategies

Strategy 2: (Student) Log Base Change

Retrieve:
Query: log base change OR how to convert \( \log_b x \) to \( \log_c x \)?

The student may use the text or symbolic query. Find sources giving the conversion rule with general bases.

Analyze: Markup sources and note down where relevant sources are located in a list (e.g., in a text file). Save some special cases like converting \( \log_{10} x \) to \( \ln x \).

Synthesize: They use this to compute \( \log_4 13 \) on a calculator as the \( \log \) button on most calculators only represents \( \log_{10}(\cdot) \).

Possible Relevant Sources: web-pages (MSE, Brilliant.org etc.), YouTube videos, online lecture slides, text documents (e.g., digital books, articles, online notes (MIT OpenCourseWare)), Online tutorial sites (Khan Academy), online answer engines (WolframAlpha), online calculators (SymboLab), online databases for definitions and theorems (ProofWiki).

Strategy 3: (Researcher) Sensitivity Conjecture

Retrieve:
Query: What is Sensitivity Conjecture? Has it been proven?

Find papers/books defining the conjecture and providing proofs.

Analyze: Since the conjecture is very technical, retrieved material is annotated with sources where terminology in the conjecture can be comprehended. An index (graph) is made capturing the chronological account of progress on the proof.

Synthesize: Results and the methods for proving this conjecture are used for similar problems, and new articles/material are created to disseminate the findings.

Possible Relevant Sources: online encyclopedias (Wikipedia, Wolfram MathWorld), online Q&A sites (MathOverflow.net, AoPS, sciencedirect.com), YouTube videos, online lecture notes, text documents (e.g., digital books, research articles), online science & math magazines (Quanta Magazine), online math databases (Cornell’s mathematics library, zbMATH Open, AMS: Math Reviews)
Retrieve: Any set $H$ of $2^{n-1} + 1$ vertices of the $n$-cube contains a vertex with at least $\sqrt{n}$ neighbors in $H$.

The search is done using a textual query with \LaTeX{} for the formulas. Related papers/books are collected and consulted for theorem definitions and proofs.

Analyze: Retrieved sources are annotated with links to other sources where terminology used can be comprehended. Highlight the name of the statement when it is found.

Synthesize: Create document summarizing the theorem name and key details, with cites/links to key sources found. Include link to a file directory on a laptop where additional notes in text and \LaTeX{} files can be found, if any.

Possible Relevant Sources: online encyclopedias (Wikipedia, Wolfram MathWorld), online Q&A sites (MathOverflow.net, AoPS, sciencedirect.com), YouTube videos, online lecture notes, text documents (e.g., digital books, research articles), online science & math magazines (Quanta Magazine), online math databases (Cornell’s mathematics library, zbMATH Open, A\&M: Math Reviews)

summarizing the key findings and where sources may be found.

As yet another example, let us consider a situation where a researcher needs to define unfamiliar notation that they encounter in a paper. As shown in Strategy 5, for this information need the researcher is simply collecting and utilizing relevant sources, rather than generating a new source of their own. If successful, they will learn that this is the Hypergeometric function $\mathcal{H}$ series, and if necessary use this name later to find identities associated with.

We also note that the interpretation of the majority of mathematical expressions is context-dependent, i.e., the same formula may refer to different concepts in different contexts. A student executing Strategy 6 will end up with multiple interpretations, which might represent:

- the distributive law: $\pi(m + n) = \pi m + \pi n$, or
- the value of the prime-counting function that counts the number of primes less than or equal to $m + n$.

The more general property of a single object signifying multiple entities is known as polysemy, such as the word ‘apple’ being used to represent both a food and a company, and often poses challenges for both information retrieval and natural language processing.
1.3. Information needs and search strategies

Strategy 5: (Researcher) Unfamiliar Notation

Retrieve:

Query: What is $\text{2F}_1(a, b; c; z)$?

The search is done as a symbolic query using \LaTeX{} for the formula. Find places where similar notation is defined and help disambiguate this specific instance.

Analyze: Retrieved material is highlighted where the same or similar notation is used.

Synthesize: Use the definition to help understand another source being read.

Possible Relevant Sources: online encyclopedias (Wikipedia, Wolfram MathWorld), online Q&A sites (MathOverflow.net, AoPS, sciencedirect.com), YouTube videos, online lecture notes, text documents (e.g., digital books, research articles), online science & math magazines (Quanta Magazine), online math databases (Cornell’s mathematics library, zbMATH Open, AMS: Math Reviews)

Strategy 6: (Student) Unfamiliar Notation

Retrieve:

Query: $\pi(m + n)$

The student does not know \LaTeX{} and so a text query is used with a unicode symbol for $\pi$.

Analyze: Highlight and make notes on where sources for the definition is found.

Synthesize: Create a web post identifying helpful sources for other students working on a group project.

Possible Relevant Sources: online resources (Wikipedia, Wolfram MathWorld), online Q&A sites (MathOverflow.net, AoPS, sciencedirect.com), YouTube videos, online lecture notes, text documents (e.g., digital books, research articles), online encyclopedia such as OEIS.
User studies and use cases. There are a small number of papers examining math retrieval online. We know of just one study examining user behaviors when using a standard text-based search engine for math (Mansouri et al., 2019b). Query logs from a general-purpose search engine in Iran were used. Compared to the general case, search sessions for math topics were typically longer with more query refinements (i.e., changing queries to try and improve results) and were less successful. Queries were also longer and more varied more than queries overall. In another interesting study, posts to threads in an online math Community Question Answering (CQA) site were studied (MathOverflow7). The authors identified patterns in the collaborative actions they exhibit (e.g., providing information, clarifying a question, revising an answer) and their impact on the final solution quality (Tausczik et al., 2014).

Earlier work considered use cases for math-aware search in a study of mathematics graduate students and faculty (Zhao et al., 2008). Surprisingly the participants did not find formula search was useful overall, perhaps because they generally knew the names of entities they wanted to search on. The study also points out that the type of a source is an important relevance factor (e.g., exercises vs. code). Another analysis of expert use cases is also available (Kohlhase and Kohlhase, 2007).

1.4 Retrieval systems

Figure 1.3 provides an overview of retrieval system interactions with people, and the specific sub-tasks from the ‘jar’ framework that they perform. Unlike the freely interacting tasks of the ‘jar’ model, retrieval systems generally perform information tasks in a fixed order, shown by arrows in Figure 1.3. The figure shows two main information flows for the collection of sources that a retrieval system uses.

1. **Index construction (offline).** Information passes from the sources at top and flows to the bottom-right, as sources are annotated with additional information, and then used to compile

---

7https://mathoverflow.net
1.4. Retrieval systems

Figure 1.3: Information Tasks in Retrieval Systems (Backend). Arrows show the flow of information. All tasks in Figure 1.2 other than Apply are shown.

2. Retrieval (online). Submitted queries are annotated and then matched against patterns in the index, returning one or more matching sources. The collection is generally consulted for passages, bibliographic data, and other contents when generating the result returned to the user.

Consulting sources.  Search engines that match queries to contents in sources are a type of filter. A standard search result is useful precisely because it contains sources with patterns of information shared with the query, omitting all other sources.

The implementation of consult tasks that access sources is important for both index construction and retrieval, and is another way that sources are filtered in a retrieval system. Source contents shown in search results directly impact our impression of which returned sources are promising. Source contents used for index construction define the available patterns
for matching queries to sources.

For example, omitting high frequency terms from queries and sources that do not signify a topic (i.e., skipping stop words such as ‘the’) can greatly reduce index sizes and increase retrieval speed, but at the risk of performing poorly on queries using these terms; a classic example is the phrase ‘to be or not to be’ from Shakespeare’s play Hamlet, which is instantly recognizable but composed entirely of stop words.

For math-aware search, a similar decision would be omitting tokens and strings representing formulas (e.g., in \LaTeX{} source files). Limitations on what can be consulted includes formulas in PDF documents, which are usually not represented explicitly (Shah et al., 2021). This and other missing information can be addressed by annotating sources.

**Annotation and indexing.** In direct contrast to filtering performed in the consult step, we will also annotate sources with additional information. This extra information can be used to add patterns for matching sources in the index, or to add information to retrieval results.

For example, some neural net-based techniques such as SPLADE automatically add words that do not appear in a source to the inverted index (Formal et al., 2021a).\(^8\) These additional terms are synonyms and other words appearing in similar contexts within a training collection. For math, a simple example is adding additional representations for formulas in sources, such as generating Content MathML for operator trees corresponding to formulas represented in \LaTeX{} or Presentation MathML, allowing formulas to be searched using both formula appearance and operation structure.

From the information obtained through consulting and annotating documents, an index of patterns for matching queries is produced. This can take different forms, but is generally one or a combination of:

1. *inverted indexes* that map entities to sources and source locations (e.g., tokens or paths in graphs for math formulas), and
2. *embedding spaces* mapping entities to points in a vector space, where entities with more similar contexts across a collection are

---

\(^8\)This augmentation is also applied to queries. Query annotations are called a *query expansion* when they add tokens or other patterns for matching additional sources in the collection.
Embedding vectors have their own dictionary mapping vectors to specific sources or source locations (e.g., when search is done on text passages or individual math formulas). This allows sources matched in vectors to be consulted when communicating results to users.

In indexes, sources are referenced using a compact reference scheme to reduce the index size. For example, we might use integer tuples such as \((\text{document}, \text{page}, \ast \text{region})\), with \(\ast \text{region}\) having a different number of integers depending on the content type. For example, for text data \(\ast \text{region}\) might provide the starting character position for a word, or the starting and ending character positions for a sentence or other text excerpt. For math formulas, formulas expressed in \LaTeX\ or MathML can be indexed similarly to text using character positions. However, for a PDF file the \(\ast \text{region}\) might instead be the formula’s location on the identified page, as given by a page number and the top-left and bottom-right \((x,y)\) coordinates for a bounding box.

**Retrieval: Querying sources and communicating system results.** We query a collection using the collection index and an annotated query containing additional terms and/or an embedding vector. Search using inverted indexes is referred to as *sparse* retrieval, while search using embedding spaces is referred to as *dense retrieval*, based on the underlying vector representations for each. In particular, term vectors representing the presence of words or formula structures in a document are mostly zeros. In general, sparse retrieval models such as BM25 that use tokens or other source contents directly for lookup are faster (Robertson and Zaragoza, 2009), but dense retrieval models such as ColBERT (Khattab and Zaharia, 2020a) are more effective (Wang et al., 2023; Giacalone et al., 2024). Some retrieval models use dense models to improve sparse models *e.g.*, SPLADE, mentioned earlier.

The improved effectiveness of dense retrieval models is partly from additional context used in defining patterns, *e.g.*, using the words referring to and surrounding a formula to represent a formula in a pattern vs. the formula alone. The use of a vector space also provides more holistic and flexible pattern matching, *e.g.*, finding source vectors with the most similar angles to a query vector, rather than matching
query formula tokens individually to vocabulary entries in an inverted index. These help bridge the vocabulary problem discussed in the next subsection.

How the final result of a query is communicated (generated) can vary substantially, and often makes use of query and source annotations. In a traditional search engine, specific sources are matched in the index for the query task, with the index comprised of some combination of inverted indexes and embedding spaces. Source contents are then used to generate a result in the communicate task, using sources and source locations matched in the index. However, for a generative question answering or retrieval system, the result of the query task may be a single vector capturing the similarity of patterns in the query to patterns within sources of a very large collection produced using a neural network. This vector is then used in the communicate task as a starting point for generating the response, for example using a second recurrent neural network trained on the collection, possibly along with additional information from the original collection of sources (e.g., with references to specific sources). Some are used to generate a list of ranked sources directly (Zeng et al., 2024), ultimately producing an extractive search result summary based on source contents.

Other recent systems such as Google’s recently released AI search assistant produce abstractive summaries of retrieved sources, which summarize matching sources but without limiting the summary to contents found in the matched sources or their annotations.

**System design.** System designers and IR researchers are interested in the efficiency and effectiveness of a retrieval system. As seen in Figure 1.3, these are observed in live systems through query and user interaction logs. For experiments, system results are computed using simulated user interactions for a fixed set of queries, and relevance scores for sources, along with a description of the information needs associated with each query. Designers and researchers also make use of additional tools for evaluation, some of which we discuss in Chapter 3.
1.5 User-system interactions and interfaces

User interfaces play a very important role in mathematical information retrieval. In addition to executing queries and returning results, how queries are entered, how results are returned, and how other information tasks in Figure 1.2 are supported can help speed up or even limit a person’s efforts to find and use information.

We next present a user-centered view of retrieval systems in math information tasks. We then share some key challenges for retrieval system interaction, along with interface designs aiming to address them.

Interfaces-in-the-task-loop. Figure 1.4 illustrates a student working to change the base of a logarithm (i.e., Strategy 2) using multiple retrieval systems. At bottom-left of Figure 1.4 is a jar holding sources the student had on hand when they started searching, new sources they find or create, along with other linked sources, e.g., by web link, citation, or mention. In addition to these sources, their queries, results from queries, and handwritten notes (e.g., from converting bases by hand) are also found in the jar. Of these available sources, the ones currently being used are at bottom-right of Figure 1.4.

Some selected sources partially or fully answer the student’s needs, but others do not, such as sources later deemed not relevant. Other selected sources might exercise knowledge such as shown for the Apply task in Figure 1.4, or come from other tasks such as annotating and indexing sources of interest. Some selected sources may be added even after finding answers, perhaps because they provide a different perspective, or have a presentation that is easier to understand.

For more complex tasks, such as for the sensitivity conjecture retrieval strategies described in Strategy 3 and Strategy 4, we may even see the focus of our selected sources drift. In the berry picking model of retrieval (Bates, 1989), people see their queries and information needs change as they search and learn. Particularly for unfamiliar topics, our needs and queries may change dramatically as our understanding does (Belkin, 1980). For example, this is likely to happen when a person explores unfamiliar concepts associated with unfamiliar notation. In our jar model, information need changes involve changing the jar lid.
Sources, Tasks, and Search

Figure 1.4: Interacting with Multiple Retrieval Systems (Frontend). Each dotted arrow represents a retrieval system backend (see Figure 1.3). Sources currently used to address the information need are shown in a separate container at bottom right.

label, perhaps using an orange sticky note placed over the original description.9

What we have in Figure 1.4 is a person generating, selecting, and using sources for needs that may change as they work. Retrieval systems are a part of this process, but not the focus.

Interaction challenges. All systems embody design decisions and biases. Naturally, no one retrieval system will be ideal for all queries or subjects (e.g., mathematics). However, users often have challenges in search that are more cognitive than system-related. These are important considerations in creating usable systems, particularly for search interfaces (Hearst, 2009; White, 2016; Holmes et al., 2019).

Norman identifies two ‘gulfs’ that limit human task performance (Norman, 1988). Broadly speaking, for retrieval systems the two main categories of interaction challenges are with expressing queries (a gulf of execution) and interpreting results (a gulf of evaluation). In both

9sticky notes: a versatile information tool in this chapter and in life.
cases, for unfamiliar topics, the user may be unable to formulate an effective query or interpret results reliably precisely because of what they don’t know, or because their understanding is incorrect (i.e., their Anomalous State of Knowledge (Belkin, 1980)).

A common cause of a gulf of execution in query formulation is the vocabulary problem, where the terms/patterns a person uses for search differ from those used to index sources. For example, in one study undergraduate students were challenged while trying to define the binomial coefficient \( \binom{n}{k} \) (Wangari et al., 2014). Because of this notation-based vocabulary problem, the students’ were unable to find a definition using standard text search. When allowed to enter the expression by hand with automatic translation to \LaTeX, they found definitions using the same text search engines.

People sometimes also encounter a gulf of evaluation when trying to identify relevant information in search results. Aside from missing relevant items in results due to the vocabulary problem, an important factor here is how retrieval results are presented to a user. For example, (Reichenbach et al., 2014) report statistically significant differences in the ability of participants to identify relevant sources in SERPs when excerpts present formulas as raw \LaTeX vs. rendered formulas. Additional gulfs occur when selected excerpts are not relevant for a need, or a person lacks the math background to understand a result.\(^{10}\)

Learning new terminology and notation while searching allows a user to extend their patterns used to express queries and identify relevant results, bridging these gulfs of execution and evaluation. Some of these new patterns might be recorded explicitly in a source, e.g., recording an unfamiliar notation for eigenvectors on a blue sticky note.

**Query input: math-aware search bars.** For the most part, math-aware search bars differ in how they include formulas. Perhaps the simplest design is for users to enter both text terms and formulas as text. An early example is the Digital Library of Mathematical Functions\(^{11}\) which accepts \LaTeX commands for formulas along with text terms in queries.

\(^{10}\)impatience, inattention, mental strain, and tiredness are also factors here.

\(^{11}\)https://dlmf.nist.gov
Figure 1.5: MathDeck query entry, formula chips, and cards (Diaz et al., 2021). Chips can be dragged, edited, and combined. Editing may be done using raw LaTeX, or a combination of operations, chips, handwriting, and LaTeX using the canvas at center. Formula cards (bottom left) contain chips, titles and descriptions. New cards can be created by users, and searched by formula & title (video: https://www.youtube.com/watch?v=XfXQhw1Qlbc).

(Miller and Youssef, 2003). The more recent Approach Zero system\(^\text{12}\) system uses MathQuill\(^\text{13}\) to render LaTeX formulas as they are typed in the search bar, and allows writing lines and argument positions to be reached with arrow keys rather than LaTeX commands (e.g., for superscripts and fraction denominators).

To avoid remembering many names for operations and symbols, or to avoid unfamiliar LaTeX or other syntax for creating formulas, usually a palette of buttons with images for symbols and operations accompanies the search bar. Buttons add formula elements including operation structures (e.g., fractions, integrals, and radicals) and symbols not found on a keyboard (e.g., greek letters such as \(\zeta\) (zeta)). Query bars with palettes often display formulas in a structured editor like those in document editors (e.g., Word). Early examples of prototypes with symbol/operation palettes include MathWebSearch (Kohlhase and Prodescu, 2013) and MIAS (Sajka et al., 2018).

\(^{12}\)https://approach0.xyz
\(^{13}\)http://mathquill.com
1.5. **User-system interactions and interfaces**

As another way to reduce the effort and expertise required for formula entry, some search bars also support *multimodal* formula entry. Multimodal query editors allow formulas to be uploaded from images or entered using handwriting in addition to standard keyboard and mouse-based entry. There are also multimodal tools such as Detexify\(^{14}\), which looks up \(\LaTeX\) commands for symbols drawn using a tablet or mouse (Kirsch, 2010). In addition to search, recognizing math in handwriting and images has been used for interactive computer algebra systems and other applications, and is an active area of research dating back to the 1960’s (Zanibbi and Blostein, 2012; Truong *et al.*, 2024).

An example of a search bar with multimodal formula entry is the MathDeck system\(^{15}\) (Diaz *et al.*, 2021). As seen at top-left in Figure 1.5, a text search box can be used to enter words and \(\LaTeX\) for formulas. Formulas can also be added from a visual formula editor shown at center-left in Figure 1.5, and using formula ‘chips’ with embedded \(\LaTeX\) (e.g., blue oval at right of the query text box). Like MathQuill and structured formula editors, MathDeck renders a formula as it is entered, but with more flexible subexpression selection and entry. MathDeck’s query and formula entry interface is designed to:

1. support text entry; natural for text, and one can easily type ‘\(x + 2\)’, or copy-and-paste \(\LaTeX\) with small changes (e.g., \(a \rightarrow x\))
2. provide symbol palettes to help enter symbols and structures
3. provide handwriting input for those who prefer it, and to avoid searching palettes for symbols & structures
4. support formula reuse in chips; chips can be used in editing, and can be exported/shared as images with \(\LaTeX\) metadata
5. construct formulas interactively using a structured editor, with larger formulas easily built up from smaller pieces.

Other multimodal query entry interfaces have similar design goals, most commonly to support image and keyboard/mouse input.

Other familiar ways to reduce query and formula entry effort are query suggestions and query autocompletion. Their helpfulness is related to the principle of **recognition over recall**: it is usually easier to recognize

\(^{14}\)http://detexify.kirelabs.org/classify.html

\(^{15}\)https://mathdeck.org
Figure 1.6: Tagent-V Formula Search Results (left) and Video Player Supporting Navigation (right) (Davila and Zanibbi, 2018). A rendered \( \LaTeX \) formula is used to search handwritten math symbols recognized in a video (Davila and Zanibbi, 2017b). Here the user has clicked on the ‘=’ of a matched formula on the whiteboard, and this advances the video to where it is first drawn (video: \url{https://www.youtube.com/watch?v=gn24qo1MLN0}).

something we see than describe the same thing from memory (Hearst, 2009). As a simple example, a query autocompletion might include a concept whose name but not formula we can remember, and allow us to quickly select a query containing both.

**Query response: communicating results.** To illustrate the communication of retrieval results, we’ll use a system for visual search that uses an inverted index. Figure 1.6 shows handwriting in math lecture videos being queried with a \( \LaTeX \)-generated formula image. The inverted index uses pairs of symbols (e.g., \((I, n)\), \( (=, A)\)) as the vocabulary for lookup. Before searching, the query image is annotated with a graph containing nodes for symbols and edges with angles between adjacent symbols.

The inverted index is queried by looking up all adjacent query symbol pairs, to find their occurrences in the video collection. Each entry in the *posting list* for a queried symbol pair (e.g., \((I, n)\)) refers to an edge connecting the same symbols drawn in a video. Before indexing, videos are annotated with keyframes of drawn symbols that overlap in time along with an adjacency graph for each keyframe. Each edge in a keyframe graph is added to the posting list for its pair of symbols, as a posting containing a unique identifier for the edge, its keyframe graph, and video. In Figure 1.6, zoomed-in video keyframe graphs are shown
in results at left, and keyframe thumbnails are shown at far right.

It is actually the drawn symbol keyframes annotated on videos that are searched. Keyframes are scored by the similarity of matched adjacency subgraphs to the query graph, based on the similarity of matched symbols (nodes) and their angles (edges). In the results shown in Figure 1.6, symbols matched in a query/keyframe have the same color, and matched graph edges are red. To avoid missing symbols due to recognition errors, symbol pairs are indexed using all combinations of possible labels for each symbol. This is how $n$ matches $M$ in the second match shown.\footnote{A variation of adding tokens to queries and documents to increase possible matches in an inverted index. Symbol similarity is computed from all label probabilities assigned to each symbol. Tangent-v has also been used to search formula collections using unique symbol labels in PDF (Davila \textit{et al.}, 2019).}

Let’s consider the results in Figure 1.6 more closely, with associated system tasks in Figure 1.3. The query result shows the top-2 matching videos, and not keyframes. To generate this view, the keyframe ranking from querying the index is restructured as a video ranking, with videos ranked by best keyframe match. Also, the annotated query and videos have been consulted to produce the graph matches shown for each video. The videos are consulted again for annotations not used for indexing: clicking the mouse on a symbol in a result keyframe makes the video player jump to where the symbol starts to be drawn.

How the search results filter and present the videos is motivated by tasks users carry out (see Figure 1.4). For example, having symbols linked to frames can help people consult videos by quickly navigating to where a formula is drawn and discussed. Results rank videos rather than keyframes to make the search results more concise and easier to consult. The communication, annotation, indexing, and querying tasks can also be supported from search results. In MathDeck formulas in results can be used directly for search, selected for editing or export, or annotated in a card with a title and description. Cards are also automatically indexed in a ‘deck’ searchable by formula or title.

Showing matching graphs in results is more helpful for designers than users; simple bolding or highlighting is more common. In contrast, MathDeck’s search results highlight matched query words and formulas
located in PDF documents (e.g., for papers from the ACL Anthology (Amador et al., 2023)).

We’ve used just two systems here to illustrate search results that rank sources, and how they interact with human information tasks. However, results from other systems have different types. Some systems plot, simplify, and/or perform requested operations on formulas, or provide solutions for math problems posed in text and/or formulas directly (e.g., using Wolfram Alpha or a math-aware chatbot). In these cases the response is an answer to a (possibly inferred) question, rather than a ranked list of sources. These are designated as question answering systems, and interactions with chatbots addressing math queries are a type of conversational search where clarifying questions and additional information may be provided or received in multiple rounds of query/result interactions.

Chatbots using Large Language Models (LLMs) have proven intriguing and useful in some instances, but there is an increasing awareness of issues related to the validity of responses and other substantive concerns (Bender et al., 2021). However, any retrieval result is only an information source – understanding and verifying any source requires additional work. Related to this, in Community Question Answering platforms (CQAs), many posts request clarification of a question, or clarify/correct posted answers and comments. This illustrates how human responses to math queries also often contain misunderstandings, ambiguities and errors.

Regardless of the result type, how information is chosen and presented in results is important. It has a real impact on the usefulness of the result as a source of information, and on how tasks other than consulting the result itself are supported. In many cases usability testing can be used to check the effectiveness of result presentations and other interface design elements, and to discover refinements and alternatives.

---

17ACL Anthology search demonstration: https://drive.google.com/file/d/1fbiMyH1fEYUJvnrhZsWzhfl0X_zo9-t/view
18Including the classic, ‘my correction to @your correction.’
19e.g., parts of the MathDeck were usability tested (Dmello, 2019a; Nishizawa, 2020; Diaz, 2021), which led to substantial improvements.
1.5. User-system interactions and interfaces

Figure 1.7: ScholarPhi system showing definition for math symbols found within the same PDF paper (Head et al., 2021). To assist skimming for details, text other than for definitions of a selected formula is greyed out. Source: https://www.youtube.com/watch?v=yYcQf-Yq8B0

Supporting tasks for individual sources. Programs used to view/consult sources can also help with the user tasks illustrated in Figure 1.4. A nice example is the ScholarPhi system shown in Figure 1.7 (Head et al., 2021). Reading formulas can be challenging, as symbols may be defined throughout a paper. ScholarPhi provides annotations decorating a selected formula/subexpression, providing symbol definitions in-place. Definitions are linked to where they appear, and text not associated with a selection is greyed out.

To produce the definition views in ScholarPhi, sources need to be annotated with formulas, symbols, and definition locations, and then definitions need to be linked with associated entities where they appear in the paper (i.e., symbols or subexpressions). Definition segmentation and linking entities are performed with natural language processing techniques. The original prototype identified math symbols with LaTeX source files used to generate PDFs, simplifying formula detection.

A second example is the keyframe list at right in Figure 1.6. When viewing a video, all keyframes for handwritten content are available in a thumbnail list. Keyframes can be selected, and individual symbols clicked on to jump to where it is first drawn in the video (similar to the search results). This requires annotating video sources with generated output.
keyframes produced using computer vision techniques.

Both ScholarPhi and Tangent-v require generating additional information using automated inference (i.e., AI), and their usefulness is limited by the accuracy and scalability of the methods employed. However, we believe that this is an important future direction for mathematical information retrieval, because the content and organization of mathematical sources can be complex. Particularly for non-expert users, mature versions of these techniques may be very helpful.

A number of well-known formats were devised or augmented to support detailed annotation with links and tags, including TIFF, PDF, and XML. Unfortunately detailed ‘semantic’ annotation has proven difficult at scale despite significant efforts. Some possible reasons include the time required to create sources before annotations, the diversity of information needs (e.g., which information do we annotate?), attaching large annotations makes files large and unwieldy, and overall progress in scalable AI has been slower than many anticipated.

As AI continues to improve, creating source annotations to support examining and navigating math sources and other user information tasks within UIs seems likely to be beneficial. Perhaps application-specific annotations such as used in ScholarPhi, Tangent-v, and MathDeck are a good starting point.

1.6 Summary

This chapter provided a brief overview of mathematical information retrieval. A simple ‘source jar’ framework capturing how people’s math information needs interact with information sources was presented. We showed how retrieval strategies can be sketched within a larger context using the framework, and how the information retrieval, analysis, and synthesis tasks in the model are performed by both people and systems, with some important differences. The chapter concludes with a characterization of challenges that people encounter interacting with retrieval systems, and examples of user interface elements designed to address those challenges for math-aware search.

In the next chapter we consider sources themselves more closely – in particular, how formulas are represented in sources.
In this chapter we focus our attention on how formulas and text are represented in sources and search indices used by retrieval systems. Figure 2.1 shows the tasks from the system overview in Figure 1.3 that we will discuss. These tasks include consulting sources for information in formulas and text, annotating formulas and text with additional information, and then constructing and organizing patterns for search in the index.

For context the query annotation and querying tasks are shown in Figure 2.1, but greyed-out. Annotations added to sources are often also added to queries, so that the correct pattern type is used to search the index. For example, for a dense retrieval model for formulas, both formulas in sources and queries are converted to vectors before identifying the formulas that are most similar to the query formula.

To help establish the roles of formulas and text and their relationships within sources, let’s consider a simple but slippery question. What does a formula tell us, exactly?
2.1 What information does a formula carry?

Let’s start addressing our question using the Inverse Document Frequency (IDF). IDF is used directly and indirectly in influential sparse retrieval models, including variants of TF-IDF (Term Frequency-Inverse Document Frequency) and BM25 (Robertson and Walker, 1994; Robertson and Zaragoza, 2009). IDF is used to score query terms that appear in sources. Its utility comes from a simple but powerful idea: query terms appearing in fewer documents are rarer and thus more specific, and so should be given higher weight when ranking sources. For example, the term ‘BM25’ is predominantly found in sources on information retrieval, while the term ‘weight’ is used for many topics and in multiple senses, including the heaviness of an object and computing scalar factors for numeric values. When scoring terms from a query containing ‘BM25’ and ‘weight’, IDF produces a higher score for ‘BM25’ reflecting its narrower usage (i.e., specialization in associated topics).

In Example 2 we see a definition for IDF, below which is the defi-
2.1. What information does a formula carry?

Formulas have visual structure representable using a Symbol Layout Tree (SLT) as shown for the \( idf \) formula. The SLT represents the placement of symbols on writing lines using the spatial relationships shown in Table 2.1.\(^1\) In the example adjacent symbols are shown using horizontal lines, and other relationships including subscript and below with directed arrows. This SLT represents the \( idf \) and \( \log \) function names in single nodes, with their characters grouped into tokens. SLTs for variables \( N, t_i, \) and \( n_i \) are subtrees, with one node for \( N \), and two nodes for \( t_i \) and \( n_i \). Table 2.1 also includes containers that represent operators and their associated writing lines compactly. We will encounter these later in the chapter.

Taken together, the excerpt is representable as a directed graph with formulas in SLTs connected to the formula nodes.\(^2\) Given this, a reader tries to recover the represented information. An important part of this process is identifying unstated or assumed information. These gaps are filled using information known to the reader, or available elsewhere in the same source or additional sources. As a simple example, a person identifies unstated information when using context to distinguish ‘\( \sin h x \)’ (sine of angle \( h x \)) from ‘\( \sinh x \)’ (hyperbolic sine of \( x \)), perhaps because of a typo. How operations and arguments are grouped needs to be correctly identified for a correct SLT representation.

In Example 2 one piece of missing information is the hierarchy of operations and arguments represented in the \( idf \) formula. While a reader is unlikely to think about this consciously, interpreting the formula essentially involves converting the SLT to an Operator Tree (OPT). In OPTs variables and other arguments appear at the leaves, with operations above the leaves in internal nodes. While SLTs are

\[ A \xrightarrow{f} B \xleftarrow{g \circ f} C \]

\(^1\)Diagrams and other graphics are frequently used in math, but outside of our discussion here; e.g., commutative diagrams can be expressed as a matrix-like SLT container, but are really directed graphs with nodes/edges labeled by formulas:

\(^2\)SLTs are hidden in the Example 2 middle panel for easier reading.
Example 2: Inverse Document Frequency (IDF)

Excerpt from (Robertson, 2004):

...Assume there are $N$ documents in the collection, and that term $t_i$ occurs in $n_i$ of them ... the measure proposed by Sparck Jones, as a weight to be applied to term $t_i$, is essentially

$$idf(t_i) = \log \frac{N}{n_i}$$

(1)

Word and formula sequence:

Assume there are $N$ documents in the collection, and that term $t_i$ occurs in $n_i$

Equation (1) operations and appearance:

Left: Operator tree (OPT) with hierarchy of operations and arguments.
Right: Symbol layout tree (SLT) placing symbols on writing lines.
oriented left-right to reflect reading order, OPTs are oriented vertically to reflect operation order. The order of operations is bottom-up in the OPT, with precedence decreasing as we move away from the leaves, e.g., ‘:=’ is applied last for the idf example.

Operations appear directly above their arguments in an operator tree. If an operator’s arguments have different roles (e.g., \( N \) and \( n_i \) in \( \frac{N}{n_i} \)) they are represented in a fixed left-right order below the operator. When argument order does not affect the result such as for ‘:=’ or ‘+’, a left-right order is chosen arbitrarily, usually reflecting the reading order in the SLT. In the example OPT, ordered operator arguments are shown with arrows, and the unordered arguments for ‘:=’ are shown with lines. A sub operator is used to represent subscripted variable names within the SLT. Orange nodes indicate where SLT symbol nodes are renamed or removed in the OPT. This includes replacing a fraction line by divide in the OPT, and removing parentheses because OPT argument edges make them redundant.

For search, it is worth considering what is fixed and what varies in OPTs and SLTs:

- In SLTs, spatial relationships are limited to those in Table 2.1, while symbols appearing on writing lines can vary greatly.
- In OPTs, an operation hierarchy with arguments at the leaves is defined. Like SLTs, operation and argument symbols vary but some operations are implied and missing in SLTs, e.g., \( xy \) represents \( x \times y \).

We can improve search by reducing variation in OPTs and SLTs, particularly when one formula has multiple representations. We will discuss this further in Section 2.2.

In addition to the underlying operations represented in formulas, there are some other types of information that formulas provide. Formulas represent or imply mathematical properties, and provide visual information in their presentation. Also, like all parts of language their uniqueness is another form of information, as the IDF formula suggests.

**Information from formulas and text.** Despite its brevity, the excerpt in Example 2 contains a fair amount of represented and implied information. But what information is represented separately by formulas and...
Table 2.1: SLT Spatial Relationships (8 total) and Containers. Relationships identify writing lines around a symbol/token. ‘Adjacent at left’ is excluded because of reading order: writing lines are read left-to-right.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inside a symbol</td>
<td>( \sqrt{N} )</td>
</tr>
<tr>
<td>Adjacent on writing line</td>
<td>( \text{idf}(t) = \log \ldots )</td>
</tr>
<tr>
<td>(at right of symbol)</td>
<td></td>
</tr>
<tr>
<td>Subscript &amp; Superscript</td>
<td>( t_i \ N^2 \int_{-\infty}^{+\infty} \sum_{i=1}^{n} )</td>
</tr>
<tr>
<td><em>(diagonally at right above/below symbol)</em></td>
<td></td>
</tr>
<tr>
<td>*Prefix Superscript &amp; Subscript</td>
<td>( \frac{n}{2}C \ ^{235}U )</td>
</tr>
<tr>
<td><em>(diagonally at left above/below symbol)</em></td>
<td></td>
</tr>
<tr>
<td>Above &amp; Below</td>
<td></td>
</tr>
<tr>
<td><em>(directly above/below symbol)</em></td>
<td></td>
</tr>
<tr>
<td>Containers: combine relationships</td>
<td></td>
</tr>
<tr>
<td>Fraction</td>
<td>( \frac{N}{n_i} )</td>
</tr>
<tr>
<td>(above + below fraction line)</td>
<td></td>
</tr>
<tr>
<td>nth-Root</td>
<td>( \sqrt{8} )</td>
</tr>
<tr>
<td>(above + inside radical symbol)</td>
<td></td>
</tr>
<tr>
<td>Matrix and Choice</td>
<td></td>
</tr>
<tr>
<td><em>(adjacent for brackets/rows, below for cols)</em></td>
<td></td>
</tr>
<tr>
<td>Note: nested SLTs within cells</td>
<td></td>
</tr>
</tbody>
</table>

\( \begin{bmatrix} 1 & 0 \\ 0 & x^2 \end{bmatrix} \begin{pmatrix} n \\ 2 \end{pmatrix} \)

*Aside from specific communities and topics (e.g., chemistry), these are rare.

text, and what information do they present together? The following is a structured summary of information in the excerpt, including uncertain details we’ll come back to later.

Variables and Functions

- Variable \( N \) refers to a quantity, similar to a noun. \( N \) acts like a common noun (e.g., street) rather than proper noun (e.g., Bath Road) because the collection is unspecified.
- Variables \( t_i \) and \( n_i \) are also common noun-like, with by a shared reference to \( i \) indicating any term in a vocabulary. For example, \( (t_i, n_i) \) could be (‘IDF’, 7) or (‘weight’, 11).
- \( \log \) represents the set of logarithmic functions with unspecified base, and so variable/common noun-like.
- \( \text{idf}(t_i) \) is a specific function applied to term \( t_i \) making \( \text{idf} \) constant and akin to a proper noun.
2.1. What information does a formula carry?

- Definitions
  - Variables in text with implied types: natural ‘counting’ numbers (\( \mathbb{N} \)) for \( N \) and \( n_i \), character strings for \( t_i \).
  - \( idf \) function in Equation (1).
  - The full definition includes textual variable definitions and the \( idf \) function definition in Equation (1).

Operations
- Division (\( \div \)), application (\( idf(\cdot) \), \( \log \cdot \)), and equivalence (\( = \)) act on variables and subexpressions (like verbs).
- Definitions
  - Missing in text and formulas of the passage.

Formulas in Text
- Variable and function names act as single-word nouns (\( N \) vs. \( Sunday \)) and noun phrases (\( t_i \) vs. ‘any given Sunday’).
- Equation (1) acts as a sentence clause with a subject (\( idf \)), subject verb (\( = \)), and predicate (the definition).

Text Only
- Provides the purpose of \( idf \): defining weights for terms.
- Provides context: formula is similar to its original proposal by Spärck Jones (Jones, 1972).

*Unspecified Details
- Base of the \( \log \) function
- Specific collection that \( N \) and \( n_i \) refer to
- Specific vocabulary (\( i.e., \) index terms/entries) \( t_i \) refers to
- Operator definitions

Notice the different communication roles that both words and formulas take in this excerpt. This includes naming/reference, actions and properties, definitions, and discourse:

1. Nouns, variables, and function names refer to objects.
2. Verbs and operators give actions on objects and object properties.
3. Definitions are provided in text and formulas, with text adding information to formulas and vice-versa.
4. Discourse: \( idf \) is referenced in the larger paper discussion, and the text provides additional information on the formula’s origin.
The excerpt also provides an example for our discussion here.

Let’s next consider an alternative definition for the *idf* function that uses no mathematical symbols or names at all:

The *inverse document frequency* for a term is defined as the number of collection documents divided by the number of documents containing the term, which is then converted to a logarithmic value.

This seems simple enough. But we lose some useful things when we remove the math notation, as summarized below.

*Compact Reference:* Variable and function name reference is more efficient than reusing descriptions, *e.g.*, referring to *N* vs. ‘the number of collection documents.’

*Visibility:* Formulas are italicized and use distinct symbols (*e.g.*, operators and greek letters), and in addition to appearing in sentences (*inline*) they may be indented and offset (*displayed*). Equation (1) in Example 2 is displayed.

*Compact Structure:* Formulas define relationships more compactly than text (*e.g.*, the *idf* formula vs. the description above). This also helps visualize mathematical concepts. In the *idf* formula, the fraction visualizes a ‘flipped’ (inverse) percentage of sources. As another example, the *distributive property* from algebra is easily expressed as \(x(y + z) = xy + xz\).

*Abstraction and Generality:* Formulas identify relationships that appear in many different contexts, *e.g.*, *idf* can be applied to formulas as well as text. Abstractions in formulas help identify, define, and name these commonalities, simplifying analysis and discussion. For example, formulas can be repurposed by redefining variables and redefining operations.

- Example 3 provides an example of applying a formula in different contexts just by redefining variables.
- We can redefine variables and operations to alter the distributive property above for vectors. To do this, multiplication \((\times)\) is replaced by \(\cdot\) (scalar product) and \(+\) is replaced by...
2.1. What information does a formula carry?

vector addition. To distinguish vectors from individual values, we’ll also make the variable names bold. This gives: \[ \mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}. \]

**Missing Information.** Having seen the helpful abstractions that math formulas provide, let’s further discuss the unspecified details for the excerpt in Example 2. It turns out that these are all *deliberate* omissions, to support abstraction in the formulas and for clearer communication:

- By not specifying a collection, \( N \) is defined for any collection.
- \( t_i \) and \( n_i \) refers to any term and the number of documents it appears in for any specific term, collection, and vocabulary.
- Logarithm values increase with input values (i.e., they are *monotonic*) so any base suffices. Omitting the base emphasizes this.
- The operators used are common in computing, and so they are assumed rather than given to save space and reader effort.

Omitting information from formulas is useful for generalization and brevity, provided that the reader is clear about what is omitted and why. The value of omission holds for text as well. When we discuss a person, place or thing, we skip shared information and unrelated details; we can find it frustrating or amusing when others forget to.\(^3\) Choosing what to omit in formulas and text is informed by:

1. The context and focus of discussion, *i.e.*, items discussed earlier and current topic
2. The background of the audience

These two factors largely dictate what can and *should* be left out.

**Rareness and topic specificity.** The IDF formula is related to *entropy* from information theory (Shannon, 1948), but is not fully compatible with it due the different probabilities involved (Robertson, 2004). Entropy measures uncertainty for events with discrete outcomes such as flipping coins or the next word or formula that will appear in a passage. It quantifies how surprising an outcome is, and equivalently how much probabilistic *information* knowing the outcome provides.

\(^3\) *e.g.*, ‘oversharing’, like telling a stranger your favorite color while booking a dentist appointment.
Example 3: A Decay Function

This is an exponential decay function for quantity $N$ after a number of timesteps $t$:

$$N(t) = N_0 e^{-\lambda t}$$

(2)

$N(t)$ is the quantity at time $t$, $N_0$ the initial quantity (when $t = 0$), and $\lambda$ is a decay constant. This same formula can be used in multiple contexts, simply by changing $N_0$ and $\lambda$. For example,

**Financial analyst:** retirement fund balance after $t$ months assuming monthly payouts and a fixed interest rate.

**Chemical engineer:** rate of a chemical reaction.

**Physicist:** discharge of the potential contained in a capacitor.

For example, we are less surprised when someone correctly predicts a coin toss than guesses the hidden top card in a shuffled deck of 52 cards. If the deck is shuffled 10 times, and after getting the first card wrong, the correct card is guessed correctly 9 times, we rightly suspect cheating because of how unlikely (surprising) this is.\footnote{With proper shuffling, 9 correct guesses in a row has an estimated likelihood of $\frac{1}{52^9} = 3.39 \times 10^{-16}$. Here the initial error was made because the signal for identifying the card had not been perfected as two adults play the game to befuddle children.}

The entropy $H$ for event $X$ with possible outcomes $\mathcal{X}$ is given by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

(2.1)

where $\log p(x)$ is the log probability of one possible outcome. These outcome values are combined in a weighted average using the likelihood for each alternative ($p(x)$). Because the maximum probability is 1.0, all outcome probabilities are smaller than standard base values such as 2 or $e$, producing negative logarithm values, and a positive entropy value after the final sum and negation. The maximum entropy for $n$ alternatives is obtained by a purely random (i.e., uniform) distribution when outcomes are equally likely with $p(x) = 1/n$, such as (fairly) guessing the top card in a shuffled deck.

Rarer terms are more informative in this probabilistic sense. The
most frequent words are articles, prepositions, pronouns/references, and other syntactic ‘latticework’ for text and their likelihood is dramatically higher than other words (e.g., the, and, your etc.), and are often referred to as stopwords. Across languages, word frequencies sorted in decreasing order follow a Zipfian distribution, with an inverse exponential-shaped curve. Rarer words and phrases in the long ‘tail’ of the curve often signify specific properties or topics (e.g., Zipf, inverse, ‘information retrieval’), but can also arise from spelling mistakes, for example.

So another form of information that formulas carry are their uniqueness or rarity within a passage or collection. For example, the \( \text{idf} \) formula is more likely to be used in sources from computer science on information retrieval, natural language processing, and data mining than other topics. In contrast, the small formulas \( x^2 \) and \( x \) will be found so frequently across disciplines that we can consider it a ‘latticework’ part of communication similar to the (i.e., a ‘stopformula’).

### 2.2 Formula representations

SLTs and OPTs are helpful abstractions when working with the appearance and operations of formulas. We use the information representable in SLTs and OPTs whenever we work directly with mathematical definitions and operations, or indirectly with applications of math. For example, we do this when using calculators, playing games, reading formulas, talking with others about what a formula represents, or when implementing a formula in a computer program.

As discussed earlier, math formulas in documents help with abstraction by capturing properties and patterns found in multiple contexts. Redefining variables and operations in formulas allows their represented information to change without altering their appearance. For familiar symbols like \( x \) or \( + \), definitions are usually omitted for brevity.

When implementing an OPT in a computer program, there are additional constraints. In code variables and operations must be uniquely defined before they can be used. This limits a flexibly-defined OPT to exactly one interpretation. Definitions can be omitted from program code when they are provided by the language or imported libraries, but all symbols must be defined for a running program to work. Definitions
must be fixed, but can be replaced, *e.g.*, by importing a different library.

Representing SLTs in code is more straightforward than for OPTs. SLTs primarily define the placement of symbols on writing lines along with formatting information (*e.g.*, spacing, fonts and sizes). The \LaTeX\ math syntax is an example of a commonly-used SLT representation. \LaTeX\ supports defining symbols and layout structures in macros and libraries.

**Representing OPTs and SLTs in code.** Example 4 provides computer code representing the OPTs and SLTs for the *idf* formula shown in Example 2 that demonstrate differences between representations for OPTs and SLTs in computer programs.

Syntax for OPTs and SLTs can both be similar to formula appearance on a page as seen for the Python and \LaTeX\ excerpts in Example 4. However, operator *prefix* syntax showing operators before arguments match OPT structure directly, as seen in the Lisp examples. To compute values from an OPT in code, we require unique definitions for operators and arguments, which require decisions about data structures and algorithms as discussed below for the implementation of subscripts.

SLT code representations are simpler than OPT code representations. They identify where and how to draw symbols, without any concern for what the symbols represent. For example, in \LaTeX\ commands such as *\sum*, *\choose*, and *\frac* helpfully suggest math operations to make the syntax easier, but the commands only define where to place symbols (*i.e.*, these are containers as shown in Table 2.1).

Let’s now consider the code examples in more detail, starting with the OPT implementations. While our example OPT can be interpreted as defining an *idf* function, for simplicity we’ll use a slightly different interpretation for testing function values. In the code we assume that:

- variables N, n, t, i, n_i and t_i have defined values,
- *idf* is defined as a function in code, and
- = (Lisp) and == (Python) test for equal values.

The Lisp and Python OPT implementations test if the *idf* function produces the value obtained by applying operations in the definition.

In the top row Lisp represents the OPT using indentation to visualize levels in the tree. All Lisp operations are grouped with their arguments
Example 4: *idf* Formula in Code (OPT & SLT)

<table>
<thead>
<tr>
<th>Lisp: prefix operations</th>
<th>Lisp + subscripts in names</th>
</tr>
</thead>
<tbody>
<tr>
<td>( = ( idf ( sub t i ) )</td>
<td>( = ( idf t i )</td>
</tr>
<tr>
<td>( log</td>
<td>( log</td>
</tr>
<tr>
<td>( / N ( sub n i )))</td>
<td>( / N n i )))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Python: infix operations + subscripts in names</th>
<th>LaTeX: appearance as SLT (writing lines)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>idf</em> ( t i ) = log( N / n i )</td>
<td>( idf( t_i ) = \log \frac{ N }{ n_i } )</td>
</tr>
</tbody>
</table>

Python and \LaTeX{} are similar but represent OPT and SLT, respectively.

using bracketed expressions, while individual variables appear without brackets. A *prefix* syntax is used, putting operations before arguments in groups. The `=, /` and `sub` operators have the form

\[
( \text{op arg1 arg2} )
\]

while the log and *idf* functions have the form

\[
( \text{op arg1} )
\]

The first Lisp expression represents the *idf* OPT using the same symbols other than replacing *divide* by the `/` Lisp operator. The operation node hierarchy is represented by nesting arguments in brackets.

Choosing how to represent the subscripts is surprisingly subtle. In the top-left Lisp expression subscripts are defined using the `sub` operator in the OPT. However, `sub` and its arguments need specific definitions if this is to run in a program. Let’s start with:

- `sub` is a standard array lookup,
- `t` is an array of strings for vocabulary terms,
- `n` is an array of counts for terms (*same* order as `t`), and
- `i` is an index position for a term.

If `i = 2`, `( sub t i )` might return ‘weight’, `( sub n i )` returns 15 as the number of documents where ‘weight’ appears, and the *idf* function receives a string argument as `( idf "weight")`. This *idf* function *must* refer to data not shown in the OPT, *e.g.*, pre-computed values in a dictionary, or a dictionary of (term, count) pairs used to compute IDF values in the function.
Now consider the top-right Lisp excerpt, where subscripts are represented in variable names. This makes the code shorter, but also less general. The code can compute values for exactly one term, rather than use the index $i$ to access any vocabulary term. Depending on other code in the program, one or the other definition may be more helpful.

An equivalent program in Python syntax is shown that includes *infix* operators, with arguments to the left and right of `==` and the division operator `/`.\(^5\) When operations appear before arguments, it is outside at left of a bracketed list. The Python code certainly *looks* closer to the original formula than the Lisp. Small differences in appearance between the original formula and Python code include the fraction being written left-right, and adding required parentheses for the log operation.

The final example is $\LaTeX$ representing an SLT for the appearance of the formula. $\LaTeX$ commands begin with a backslash, and the fraction is represented using the `\frac{}{}` command. Only the parentheses (`(' and `)') that actually appear in the drawn formula are included in the $\LaTeX$. Subscripts are represented with operators to capture writing line changes, with `_' being the $\LaTeX$ subscript operator rather than part of variable names. There are commands in $\LaTeX$ that modify the interpretation of `_' and the superscript operator `^`. For example, the `\displaystyle` command changes argument positions from sub/superscript to above and below for $\sum_{i=1}^n$ in Table 2.1.

**Canonicalization and MathML.** Example 5 visualizes some common ways to identify properties of variables and operations in OPTs. At middle we number each unique variable from left-to-right in the OPT, starting from 1. This type of numbering for entities is known as an *enumeration*. `2` for $i$ is repeated because it appears twice in the expression.

We can use enumerations to capture variable placement while ignoring specific variable names. For example, using variable enumeration the *pythagorean theorem* expressed as

\[
x^2 + y^2 = z^2
\]

and

\[
a^2 + b^2 = c^2
\]

\(^5\)Technically, \texttt{log} should be \texttt{math.log}.\]
Example 5: OPT Variable Enumeration and Symbol Types

Both have the form

\[ 1^2 + 2^2 = 3^2. \] (2.4)

Enumerations are helpful for unifying the variables, making the two versions of the pythagorean theorem identical. We do this by substituting variables (e.g., \( \{x/a, b/y, c/z\} \)) or representing variables with enumeration numbers instead of variable names. Unification is useful in many contexts, including theorem proving and formula search. It is possible to enumerate operations in the internal nodes as well, but this is less common for retrieval applications.

Variable enumeration removes distinctions between formulas with identical operations but different variables. Applying transformations like this that generalize formulas and reduce the number of unique formulas is known as canonicalization. Canonicalization often involves normalizing symbols and structures as well. A common normalization is re-ordering variable names alphabetically for operations where the order of arguments is unimportant, e.g., to have both \( x + y \) and \( y + x \) represented by \( x + y \).

Both OPTs and SLTs may be canonicalized using enumeration, normalizations, and other transformations. Without canonicalization it is harder to identify the same information in different formulas. However, using too much canonicalization can remove meaningful differences between formulas. A common compromise is to use multiple formula representations. For example, we might search OPTs using enumerated variables, and then refer back to the original OPTs to rank those
Types are also often used for annotation and canonicalization, as shown at right in Example 5. Here we have an OPT with nodes replaced by an assigned type for each symbol. I indicates identifiers (i.e., names) for variables and operations/functions like the idf function identifier. Note that the identifier idf must represent an operation: it appears at an internal node and not a leaf in the OPT. In our example, other predefined mathematical operations are assigned one of two types: O for operations with ordered arguments (sub, log, divide) and U for operations with unordered arguments (=). Among other uses, types can be used as constraints when unifying symbols including operators with the same argument structure in formulas.

There are many formula file encodings available. Here we focus on the XML-based Mathematical Markup Language (MathML\textsuperscript{6}), which has been used in a number of evaluation benchmarks for math-aware search. Being XML-based, the syntax for representing formulas is much closer in structure to the prefix syntax of Lisp than the Python syntax with infix operations in Example 4. All MathML commands have a start and end tag, each containing a list of tags. The syntax is roughly

\[
<\text{tag-command} [\text{attribute list}]>
<\text{argument-1}>
\ldots
\end{\text{argument-1}}
\ldots
<\text{argument-n}>
\ldots
\end{\text{argument-n}}
</\text{tag-command}>
\]

This is used both for SLT representation in Presentation MathML, and OPT representation in Content MathML. For single symbols tags can be combined with the symbol itself, e.g., for the $\log$ operator in Content MathML.

Some MathML commands, such as $\text{msub}$ for subscripts in OPTs and SLTs have a fixed number of arguments. Others like $\text{apply}$ in Content MathML and $\text{math}$ or $\text{mrow}$ representing writing lines in

\textsuperscript{6}https://www.w3.org/Math
Example 6: MathML for IDF Formula (LaTeXML Output)

The MathML shown was produced by LaTeXML\(^{a}\) using the LaTeXML in Example 4. idf is not defined; \(i\), \(d\), and \(f\) are treated as variables.

Symbol Layout Tree in \textit{Presentation MathML}

\[
idf(t_i) = \log \frac{N}{n_i}
\]

Other than adjacency, relationships are represented in tags (\textit{e.g.}, \texttt{<msub>} for subscript, \texttt{<frac>} for fraction). Unicode \texttt{x2061} is inserted by LaTeXML representing the \textit{function application} of \texttt{log} to \texttt{N/n_i}.

Operator Tree in \textit{Content MathML}

\[
\times \times \times \div \div \frac{N}{n_i}
\]

Nested \texttt{<apply>} tags represent the nesting of operations in the OPT. Each \texttt{<apply>} tag contains a function (\textit{e.g.}, \texttt{log}) or relation (\textit{e.g.}, \texttt{=}) and its arguments. With \texttt{idf} interpreted as three variables, a multiplication replaces the application of \texttt{idf} to \(t_i\).

\(^{a}\)https://math.nist.gov/~BMiller/LaTeXML
Presentation MathML have variable length argument lists. The apply operation is close to the bracketed operation groups in Lisp: the first argument is an operation followed by its arguments.

MathML also defines types for arguments, including <mi> and <ci> for variable identifiers, <mn> and <cn> for numbers, and <mi> and <mo> for operators in SLTs/Presentation MathML (e.g., <mi>log</mi>). Defined operations in OPTs/Content MathML have their own predefined tags, and so log appears as <log/> and not <ci>log</ci>. The LATEXML tool used to produce Example 6 is aware that \log is an operator, and inserts an invisible node in the Presentation MathML to capture the application to the fraction, using the Unicode value x2061 in hexadecimal. This symbol does not appear when this formula is rendered (e.g., in a web page by MathJax7).

idf was not defined as an operation when translating the LATEX in Example 4 to MathML. As a result this is broken up into three adjacent variables in the SLT, i, d, and f. In both the Presentation (SLT) and Content (OPT) MathML excerpts, these are treated as variable identifiers. Because of this change, in the Content MathML the letters are multiplied with each other and ti.

Because a fixed set of definitions is required to convert formulas, for large collections inconsistencies such as those seen in Example 4 are common. There is a Content MathML <cerror> tag for unrecognized symbols or structures seen frequently in large-scale conversions as well.8 Fortunately, for search if these interpretations not intended by their authors and ‘errors’ are consistent, they still provide patterns that are useful in search, provided that both formulas in queries and in sources are converted in the same way.

Getting back to canonicalization, we often restructure automatically generated OPTs and SLTs in MathML before indexing to simplify the tag structures. Common examples include removing <cerror> tags, and replacing nested <mrow> tags by a single <mrow> or top-level <math> tag in Presentation MathML, e.g., to avoid:

    <mrow>i <mrow>d <mrow>f ... </mrow></mrow></mrow>

---

7 https://www.mathjax.org/
8 Of course, some formulas are actually incorrect.
2.2. Formula representations

Similarly, for Content MathML it is also common to ‘flatten’ some operation sequences to a single operation node. For example, we can flatten multiplications to a single multiplication (represented by $\text{times}$) without changing the original meaning, because multiplication is an unordered operation (e.g., $a \times b \times c = c \times b \times a$). Both of these transformations for $\text{mrow}$ and $\text{times}$ are applied in Example 6, flattening a chain of $\text{times}$ nodes into one node in the Content MathML (OPT), and removing $\text{mrow}$ tags entirely from the Presentation MathML (SLT).

Other formula representations. SLTs and OPTs are probably the most natural representations for formulas, giving appearance by symbol placement and the represented operation hierarchy. However, in many source types including images, videos, and PDF files, the location of formulas are unavailable, as well as SLTs or OPTs. In PDF formulas are usually represented in embedded pixel images (i.e., raster images) or PDF drawing instructions for characters and lines (i.e., vector graphics, such as in SVG or PDF-rendered $\LaTeX$).

For these source types, we need to annotate sources with formula locations and/or create an isolated formula collection. Once formulas have been located or placed in individual images, we need to either (1) annotate formulas with SLTs/OPTs using recognition systems, (2) embed formula images directly as vectors for dense retrieval, or (3) create visual representations that can be computed for raster (e.g., PNG) or vector (e.g., PDF) images. Image embeddings and visual formula representations provide patterns distinct from SLTs. In this regard they can be viewed as complementary rather than strictly as substitutes or replacements for SLTs.

Both formula detection and recognition are active research topics in the document recognition community, and image embedding and image retrieval continue to be explored extensively in the information retrieval, computer vision, medical computing, and multimedia research literatures. Here we will consider two examples of visual-spatial representations that have been used as alternatives to SLTs for formula search; others are certainly possible, and an interesting direction for future work.
XY-Cut Trees

### Pyramidal Histogram of Characters (PHOC)

**Level 1**

\[
y = \frac{1}{2} x + \sqrt{9}
\]

'\(y\)': [1] \(\cdots\) '9': [1]

**Level 2**

\[
y = \frac{1}{2} x + \sqrt{9} \quad y = \frac{1}{2} x + \sqrt{9} \quad y = \frac{1}{2} x + \sqrt{9}
\]

'\(y\)': [1,1] \(\cdots\) '9': [1,1] \(\quad\)

**Level 3**

\[
y = \frac{1}{2} x + \sqrt{9} \quad y = \frac{1}{2} x + \sqrt{9} \quad y = \frac{1}{2} x + \sqrt{9}
\]

'\(y\)': [0,1,1] \(\cdots\) '9': [1,1,1] \(\quad\)

XY-Cut Trees

<table>
<thead>
<tr>
<th>Recursive X-Y Tree</th>
<th>Standard X-Y Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>(c)</td>
</tr>
</tbody>
</table>

(b) Recursive X-Y Tree

(c) Standard X-Y Tree
2.2. Formula representations

For vector images with explicit symbol locations (e.g., in SVG or PDF) or where isolated symbols in images have been recognized, we can use a spatial formula representation. We saw an example of this with the line-of-sight graphs over symbols used to search handwritten and typeset math in the previous chapter. An example of a region-based spatial representation is the Pyramidal Histogram of Characters (PHOC), which represents symbols in a fixed set of recursively partitioned regions. PHOC was originally created for retrieving words in handwritten text (Almazán et al., 2014) but can be generalized in a straight-forward way for representing two-dimensional structures like formulas (Langsenkamp et al., 2022; Avenoso et al., 2021; Amador et al., 2023). For a formula, the whole formula is the first region. Additional regions are defined by splitting this formula region into 2, 3, or more regions horizontally, vertically, and concentrically using ellipses or rectangles. A bit vector is created for each symbol, using a ‘1’ to mark each region that the symbol appears in. Each formula is then represented by a very long vector that concatenates this bit vector for every symbol in the vocabulary. A much more compact representation defines vectors only for symbols appearing in a formula, as illustrated in Example 7.

For raster (pixel-based) images symbol locations are unknown, but we can still capture visual structure without OCR using an XY-cut tree. XY-cut trees partition touching pixel groups (connected components) by recursively cutting at vertical and horizontal gaps in an image, either in strict alternation or using the largest gap in either direction (Ha et al., 1995; Nagy and Seth, 1984). The resulting graphs are related to SLTs: cutting vertically tends to partition the image at subexpressions along a writing line, and cutting horizontally tends to separate writing lines, e.g., above and below a fraction. Symbols can be recognized or features computed from sub-images at nodes (Zanibbi and Yu, 2011). This was one of the earliest spatial representations used for OCR of math formulas (Okamoto and Miao, 1991). XY-trees could also be used when symbols are known (e.g., for PDF), as it is simply a region partitioning method.

---

9Some additional steps are needed to remove radicals to access their subexpressions, and to identify where writing lines are subscripted or superscripted.
There are also alternative symbolic representations that can be used for search, such as substitution indexing trees originally developed to support automated theorem proving (Graf, 1995b). Substitution indexing trees can be used to group OPTs or SLTs with shared structure using a hierarchy of symbol and operation replacements and enumerated variables (Kohlhase and Sucan, 2006; Schellenberg et al., 2012). These trees represent a set of possible operation sequences that produce a concrete formula at the leaves, with operations performing replacements top-down in an OPT.

Whatever formula representations are used, it is common to embed them in vectors for use in dense retrieval models. Generally speaking, formula embedding vectors are obtained by translating an image, text, or graph into a fixed-length vector. Formula images may be converted to vectors, many methods convert \LaTeX{} and/or MathML token sequences to vectors, and some convert SLT and/or OPT graphs directly to vectors using Graph Neural Networks (GNN). For example, a graph convolutional neural network can be applied to SLTs in Presentation MathML generated from \LaTeX{} formulas in arXiv papers (Pfahler and Morik, 2020).

### 2.3 Formula annotation and indexing

Figure 2.1 illustrates the main tasks for indexing sources for retrieval. When we talk about indexing, we’re actually referring to a process that consults and annotates sources with additional information, and then creates a collection index. The collection index contains data structures used to search the collection of sources using patterns generated from sources and their annotations. More concretely, indexing involves:

1. consulting source text, formulas and sub-expressions and generating dictionaries for fast lookup and analysis (i.e., source content annotations),
2. adding new information to sources using additional dictionaries, e.g., formula locations, formula representations, connecting formulas to descriptions and variables to types (i.e., additional source annotations), and
3. generating inverted index and/or dense vector index files from
source contents and additional annotations.

In this section we first discuss indexing formulas in isolation, and then how text annotations can be added and included in indexing. We’ll then briefly talk about indexing sources as whole, including all formulas and text in the next section.

**Preliminary book-keeping: sources and formulas.** Whatever method we use for formula search, retrieval systems need to access (1) individual sources and (2) the location of formulas within sources. We need to detect formula locations in sources if they are not explicitly identified. For example, videos and PDF documents generally do not identify formula locations while \LaTeX and MathML demarcate formulas using known symbols and tags.

We generally use integer identifiers for sources and formulas because of their fixed small size. These integer identifiers are faster to process than variable-length character strings, and their use avoids duplicating sources in the index. For example, given a collection of \LaTeX documents we might produce two dictionaries:

1. **Source formula locations**
   \[
   \text{formula-int-id} \rightarrow (\text{source-int-id}, \text{start-char-index}, \text{end-char-index})
   \]

2. **Files for sources**
   \[
   \text{source-int-id} \rightarrow \text{file-location}
   \]

To generate a search result containing \LaTeX for a formula, we use three lookups. The first uses the formula id to obtain the source and excerpt containing the formula, the second retrieves the file location using the source id, and the third obtains \LaTeX commands from the excerpt within the file after it is loaded. The dictionaries save space through *indirection* (i.e., ids referring to ids). For example, within the index the file names for sources only appear in the second dictionary – all other references are source ids.

Where formula locations are detected, a modified formula location dictionary is needed. An example are the Page-Region-Object tables used for ACL anthology PDFs in the MathDeck system (Amador et al., 2023). Each detected formula is assigned an identifier and annotated with its source, page, and page location.
1. *Detected PDF source formula locations*

   \[ \text{formula-int-id} \rightarrow (\text{source-int-id}, \text{page-int-id}, \text{page-region-bb}, \text{obj-int-id-list}) \]

- \text{page-region-bb} is two \((x, y)\) coordinates giving top-left and bottom-right corners for a page region \((i.e., \text{a bounding box})\). A secondary dictionary holds the object type, value, and bounding box for each \text{obj-int-id}:

   *Formula object index*

   \[ \text{obj-int-id} \rightarrow (\text{type}, \text{value}, \text{obj-region-bb}) \]

For raster images, formulas have only one object, the raster image. For formulas containing drawing instructions, there may be multiple objects and types \((e.g., \text{character/line/image})\) and values \((e.g., \text{character/line endpoints/raster matrix})\).

So far our formula dictionaries represent only the locations of formulas in sources. This means that formula representations must be retrieved from files. It is convenient to add additional dictionaries for formula representations, ideally with manageable added space. We will also need to annotate sources with representations we want not provided in the sources, \(e.g., \text{OPT}\) is normally not included in \LaTeX\ files or HTML pages. For example, with three representations, we could produce one dictionary for each and use a single formula id to get each representation separately:

3. **SLT id:** \text{formula-int-id} \rightarrow \text{SLT-int-uid}

4. **OPT id:** \text{formula-int-id} \rightarrow \text{OPT-int-uid}

5. **\LaTeX\ id:** \text{formula-int-id} \rightarrow \text{\LaTeX-int-uid}

Notice that each dictionary entry is another identifier; here \text{-uid} represents \text{unique identifier}. These refer to three more dictionaries (dictionaries 6-8) holding \text{unique} SLT, OPT, and \LaTeX\ representations, \(e.g., \text{\LaTeX-int-uid} \rightarrow \text{latex-string}\). Unique formulas are identified after canonicalization to ignore unimportant differences \((e.g., \LaTeX\ comments)\). This avoids repeating common expressions like \(x^2\) thousands or millions of times. We can now retrieve a representation by formula id in two lookups: first for the unique id, and then for the representation \((e.g., \LaTeX\ string)\). We can also produce formula dictionaries for additional representation types and/or canonicalization methods.
The choice of how we canonicalize formula representations is important, as this impacts which formulas are treated as identical, and how abstract/generalized formula representations are (e.g., from variable enumeration and token type information). After this we can produce the building-block dictionaries for sources and formulas such as those above, which provide the foundation for formula search index construction.

**Indexing formula sub-structures.** We can also index parts of formulas. Example 8 show paths for OPTs and SLTs of different lengths. For the OPT the paths are rooted to variables $t$ and $i$ in leaf nodes, ending at the root of the OPT. This guarantees that the indexed paths belong to complete subexpressions in the OPT. For the SLT, we use a more generic approach, constructing all paths of lengths 1 and 2. Other approaches to generating paths or substructures have also been used for SLTs and OPTs, e.g., complete subtrees rather than paths.

Let’s again use dictionaries to assign an identifier to each unique path seen in our formula collection, representing paths as token tuples (e.g., $(t, \text{sub})$ with unique id 7). The next two dictionaries define a *vocabulary* of paths for SLTs and OPTs in our collection.

9. **SLT path vocab:** SLT-path-tuple → SLT-path-int-uid
10. **OPT path vocab:** OPT-path-tuple → OPT-path-int-uid

We then create two more dictionaries for posting lists identifying unique SLTs and OPTs containing a path.

11. **SLTs with path:** SLT-path-int-uid → [ SLT-int-uid, ... ]
12. **OPTs with path:** OPT-path-int-uid → [ OPT-int-uid, ... ]

With these new dictionaries we can identify all formulas whose SLT contains a specific path tuple with 3 lookups. The first returns the unique id for the SLT tuple, and the second gets a list of SLT-int-uid for unique SLTs. We then do reverse lookups for each SLT-int-uid in dictionary 3 to collect all matching formulas from their formula-int-id. OPT paths are handled in the same manner. This organization allows us to quickly access unique representations separately from all formulas with duplicates, while still connecting unique formula representations with all of their occurrences in a collection.

As a simpler example, to index LATEX tokens, we just perform text indexing at the formula level. Just as for text indexing, we produce a
Example 8: Indexing Paths in SLT and OPT

**Operator Tree Paths:**

**Symbol Layout Tree Paths:**

**SLT Path Length 1 (All Edges)**

**SLT Path Length 2 (Partial Edge List)**
2.3. Formula annotation and indexing

unique integer id for each token – equivalently a token tuple of length 1.
We can then map each token to a posting list of unique \LaTeX formulas:

14. \LaTeX token vocab: $\LaTeX$-token $\rightarrow$ $\LaTeX$-token-int-uid

15. \LaTeX with token: $\LaTeX$-token-int-uid $\rightarrow$ [$\LaTeX$-int-uid, ...]

We can do the same type of 3-stage lookup used for SLT paths above to find formulas containing a token, using a reverse lookup on dictionary 5 in the final step. For larger retrieval units than single tokens, it is generally better to use SLT paths rather than \LaTeX code excerpts, because this captures writing line structure more flexibly.

Retrieval unit granularities and index types. Along with the formula representations and canonicalization, the size or granularity of the unit(s) used in retrieval is very important. Intuitively, patterns for whole formulas represent more information, while formula parts represent specific information. Which retrieval unit is more helpful depends upon the query type, how units are represented as patterns, and how similarity is measured. Similarity in sparse vector spaces is computed by accumulating weighted matches from inverted index lookups, e.g., using IDF (Zobel and Moffat, 2006). Vector similarity is instead geometric, and usually computed from angles between vectors (e.g., cosine similarity) or a line between vector endpoints (e.g., Euclidean distance).

For example, for sparse retrieval we can use our unique SLT representation dictionary and a second dictionary mapping canonicalized SLTs to unique SLT ids, i.e., \LaTeX-canon $\rightarrow$ \LaTeX-int-uid (i.e., a unique SLT vocabulary). Unfortunately, querying returns only formulas with the same unique SLT id; results are empty if the formula is not in the vocabulary. This will perform poorly if we want to find variations of the idf formula using a formula query, for example. However, searching using SLT paths as described above matches formulas different from the query formula, because we search multiple paths for a single query, and each path need only match part of a formula. Because sparse retrieval is essentially dictionary lookup, approximate matching for query lookups

\footnote{Sparse retrieval is generally faster because query patterns are looked up in dictionaries, rather than embedded by a neural net followed by finding neighbors in the embedding space.}
must come from either canonicalization (e.g., variable enumeration), or query expansion.

The choice of pattern(s) to embed in vectors for dense retrieval is just as important as for sparse retrieval. For example, imagine that we embed formula SLTs in vectors and store these in a matrix (i.e., our index of ‘dense’ vectors). A query SLT vector will return many results sorted by increasing distance from the query vector. However, many similar formulas may not be \( idf \)-related because the query pattern represents the formula as a whole. Including smaller patterns such as paths that include the function name \( idf \) will be helpful. For dense retrieval canonicalization is still often helpful e.g., normalizing \(<mrow>\) tags in Presentation MathML to avoid false distinctions.

There is a clear difference in purpose here with associated trade-offs: inverted indexes catalogue the exact locations of specific patterns in formula representations, while embedded spaces represent similarity through geometric differences between transformed (vectorized) patterns. Construction-wise, inverted indexes are obtained through dictionaries such as described above, where ultimately the key for each index is some type of integer (e.g., for source, formula, unique formula, or unique path). We often use dictionaries when producing dense indexes as well, including enumerating pattern vocabularies for use in indexing (e.g., paths or token integer ids), and to map embedding vectors to their original formula/source.

**Constructing embedding spaces for dense retrieval.** Unlike the graph or text-based formula representations organized within inverted indexes, for dense retrieval these formula patterns are transformed into embedding vectors. A standard approach is having a neural network play an imitation game that is a classification task. In the game we hide tokens (i.e., mask them) in an input sequence (e.g., \( \LaTeX \)), or node/edge labels in an input graph (e.g., an OPT or SLT), and the network produces likelihoods for every alternative in a vocabulary. During training, this game is played repeatedly for multiple iterations over a training data set, with network weights updated to improve estimates as defined by
2.3. Formula annotation and indexing

a loss function that scores the game, such as the cross-entropy loss.\textsuperscript{11} Masking and other types of self-supervised learning capture a language model reflecting how objects are associated through their likelihood of appearing together and in particular ways. The often large amounts of computation required is somewhat confusingly referred to as pre-training because network weights are not optimized for retrieval directly.\textsuperscript{12}

To further improve dense retrieval performance, additional imitation games known as learning to rank tasks are run using the network that require ranking two or more items at time. Network output layers are replaced to play a new game. Weights are again updated to try and improve a chosen metric; as a simple example, for ranking two items at a time this might be the cross-entropy loss with four alternatives: (1) item 1 before item 2, and (2) item 2 before item 1, (3) equally relevant, (4) both non-relevant. This is more generically know as fine-tuning of network weights. Learning to rank requires test collections where sources relevant to specific queries have been identified, as described in the next chapter; normally relevance labeling is at most partially automated (Faggioli \textit{et al.}, 2024). As a result, data available for learning-to-rank is often much smaller than for language model learning. Data for pre-training is often created by randomly hiding labels or applying other manipulations without human involvement. After training is finished, output layers for last game played are removed, and the remaining network layers kept for use in embedding formulas.

Some final steps include computing and storing embedding vectors for collection formulas in a matrix, along with additional matrices or dictionaries storing references to our previously built dictionary keys, \textit{e.g.}, formula-int-id values for individual formulas in sources, or OPT-path-int-uid for unique OPT paths in the collection. Multiple

\textsuperscript{11}Related to our earlier discussion of IDF and entropy, the cross-entropy loss $\ell_{CE}$ is interesting. For hidden label $l_x$ with correct label $x$ we have $\ell_{CE}(l_x) = -\log p(x)$, where $p(x)$ is the correct label probability estimate. This is exactly the same quantity for alternatives seen in the entropy equation (Equation 2.1). Note that small correct label probabilities produce very large losses.

\textsuperscript{12}Pre-trained' language models often produce surprisingly strong retrieval baselines. For example, symbols or sub-expressions that look quite different may have similar vectors if they are often used in similar contexts, because when masking these, one or the other will be more likely than other alternatives.
Excerpt from (Robertson, 2004):...

\[ \text{idf}(t_i) = \log \frac{N}{n_i} \tag{1} \]

- Individual variable and function name entities indicated by filled colored boxes
- Entity descriptions shown with underlines matching entity box color
- The anaphora of them referring to N documents shown with an unfilled blue box

Dense indexes can be created for different representations, canonicalizations, and retrieval units (e.g., formula vs. path vs. LaTeX token).

**Textual Formula Annotations.** As discussed at the beginning of this chapter, formulas provide useful abstraction through the ability to redefine their symbols. However, this means that the symbols of a formula in isolation are often ambiguous, as well as what specifically a formula represents. We can address this by contextualizing the formulas through annotating them with references to surrounding text and formulas. We previously saw a visualization of this in Figure 1.7, where symbol definitions are shown directly around the symbols of a formula in a paper.

Math Entity Linking (MEL) connects math formulas to surrounding context, and includes the related task of resolving *co-references* (i.e., capturing multiple references to the same entity). Context may include descriptions for symbols and entire formulas, other formulas defining symbols in a formula, and external sources (e.g., linking formulas to Wikipedia pages). Example 9 shows a possible annotation of Equation 1 in the Example 2 excerpt. Aside from annotating symbols with descriptions, by resolving the *anaphoric* reference from ‘them’ to ‘N documents’, we can include text references identifying that \( n_i \) represents a document count for term \( t_i \), and that the *idf* formula is a ‘weight to be applied to the term \( t_i \).’ Beyond this small example, we
should note that symbols like $x$ and $\lambda$ are frequently re-defined within a single paper, leading to multiple definitions (Asakura et al., 2022). This complicates the task of coreference resolution, where multiple references to the same mathematical symbol or entity need to be identified and disambiguated when symbols are redefined (Ito et al., 2017). \(^{13}\)

The majority of early work on MEL was rule-based approaches, due to limited data for training machine learning models. One of the earliest textual formula annotations linked math expressions to their corresponding Wikipedia page (Kristianto et al., 2016b); unfortunately not all math expressions have Wikipedia pages, and context provided in the document where a formula appears is likely more relevant and/or accurate for the formula. Another early system annotated formulas with descriptions and relationships to other formulas in dependency graphs (Kristianto et al., 2017). Textual descriptions are extracted using an SVM-based model to link description nodes to formulas and symbols (Kristianto, Aizawa, et al., 2014). References between formula nodes are captured through structural matching of formula sub-expressions.

Later systems including MathAlign (Alexeeva et al., 2020) focused on textual annotations within the documents where formulas appear. There has also been work on automated variable typing, where pre-defined mathematical types (e.g., integer, real) are assigned to variables in mathematical formulas using sentences containing descriptions where a symbol appears (Stathopoulos et al., 2018).

More recently there was the SymLink shared task at SemEval 2022 (Lai et al., 2022). Symlink requires extracting math symbols with their textual descriptions from \texttt{\LaTeX} source files collected from the arXiv. The main task requires this to be performed within a \texttt{\LaTeX} paragraph. First, all text spans (contiguous excerpts) containing math symbols and descriptions are identified, and then symbols are matched with their descriptions. The dataset provides more than 31,000 entities and 20,000 relation pairs. This available data allowed several BERT-based models to be proposed for the task.

As with everything to be used for retrieval, these textual descrip-

\(^{13}\)The Math Identifier-oriented Grounding Annotation Tool (MioGatto) (Asakura et al., 2021) provides a tool for annotating different roles for formulas and symbol, linking identifiers to pre-defined math concepts extracted from the document.
Representing Formulas and Text

Formulas and their links with formulas and formula symbols need to be compiled in tables. As a simple example, we might represent formula text descriptions using source and excerpt range (span) information, as we did for formulas in LATEX files earlier, and where specific formula elements (objects) are described by text and/or another formula:

- **Formula annotations**: formula-int-id → text-span-id
- **Text spans**: text-span-id → (source-int-id, start-char-index, end-char-index)
- **Symbol annotations**: obj-int-id → (text-span-id, formula-int-id)

If needed, representing formulas within text spans can be done using minor changes to dictionary entries, and the types of formula dictionaries described previously. From here we can produce inverted indices using these linked dictionaries directly, or using smaller parts (e.g., cutting up long text descriptions). For dense retrieval models, we use one or more of the resulting pattern indexes, e.g., embedding symbols/formulas with their associated descriptions, and separately creating a dense vector index for textual descriptions on their own.

Aside from annotation and indexing for retrieval, textual formula annotations can be used for other downstream information tasks. For example, using formula symbol identifier descriptions as features for automatically generating Mathematics Subject Classification (MSC) subject codes (Schubotz et al., 2016). MSC is a collaboratively-produced hierarchical classification scheme used to identify subject codes for papers in math journals. Recent math-aware search engines have also explored using annotated formulas as their collection, including the math entity cards in MathDeck (Dmello, 2019b) that connect formulas to titles and descriptions from Wikipedia. Another retrieval system, MathMex (Durgin et al., 2024) indexes formulas that appear with their textual descriptions in a document.

### 2.4 Indexing formulas and text

Having discussed indexing formulas and their annotations, let’s now briefly discuss indexing formulas and text together. Example 10 presents a definition of IDF from Wikipedia. Notice that our definition has not
changed in any significant way, however, the formula used to represent idf has changed. The source collection now appears explicitly as a parameter $D$, and we no longer use $i$ to refer to any term in a vocabulary. The number of documents containing a term, previously variable $n_i$ is now an expression for the number of documents containing the term, with the term noted as simply $t$, and not $t_i$. Here is an example of a different formula for the same idf concept, with different variables and operations.

Consider the HTML excerpt in Example 10, with some parts hidden by [...]. We have a section title (<h3>), paragraph (<p>) and an SLT formula given as Presentation MathML with the original \LaTeX\ used to generate this given as a <math> tag attribute and <annotation> tag value. The tag <semantics> can be is misleading here – this signifies an SLT definition rather than an OPT definition.

We can apply the same types of formula annotations discussed earlier, e.g., generating additional formula representations, indexes for parts of the formula, connecting text descriptions to symbols, and so on. However, we also have opportunities to break the text into different retrieval unit patterns. Titles often identify the topic of a text section, and we can index this separately to provide search of title specifically, or to support weighting titles higher in retrieval scores. Also, we have a link to the Logarithmic scale article, and we can use the article name to annotate text in the tag, and/or follow the link to annotate the tagged phrase with text from that article.\footnote{This is riskier, as there will be a context shift when moving to the cited article.}

To compile content for indexing, it is often helpful to index formulas first, and then replace formulas by their generated identifiers in text passages. For example, the MathML excerpt could be added to a Presentation MathML formula index, and we could then replace the formula tags with the index identifier, e.g., <math [...] </math> becomes EQ::42 for formula-int-id 42. If instead want to use the HTML directly including the MathML tags for indexing, we can use an approach similar to that for \LaTeX\ discussed earlier. A dictionary is used to assign integers to unique tokens, and then a secondary dictionary represents each file as a token integer sequence.\footnote{Token integer sequence representations are common for training neural networks}
Example 10: IDF Definition from Wikipedia tf-idf Page

### Inverse document frequency

The **inverse document frequency** is a measure of how much information the word provides, i.e., how common or rare it is across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

\[
\text{idf}(t, D) = \log \frac{N}{|\{d : d \in D \text{ and } t \in d\}|}
\]

---

**HTML with Presentation MathML + \(\text{TeX}\)**

```html
<h3 id="Inverse_document_frequency">Inverse document frequency</h3>

The <b>inverse document frequency</b> is a measure of how much information the word provides, i.e., how common or rare it is across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

\[
\text{idf}(t, D) = \log \frac{N}{|\{d : d \in D \text{ and } t \in d\}|}
\]

```
In contrast to treating formulas as token sequences, we can also convert text to graphs that contain formula graphs (e.g., OPTs or SLTs). An Abstract Meaning Representation (AMR) graph represents the meaning of text using a hierarchical representation of subject, objects, actions, and attributes (Langkilde and Knight, 1998). An example is shown in Example 11 for a query that includes a formula. AMR graphs have a very similar purpose and even structure as OPTs used to represent operations in formulas. For example, the root of the example AMR is a verb with two arguments and a mode modifier (imperative) indicating that the statement is a command or request, with one argument for who receives the command (you) and what is requested (a thing that is the general solution to the provided equation). The AMR representation at left was produced by a neural network after replacing the formula by the identifier $\text{EQ:10}$ in the input text. This allows the network to treat the formula as a single node in the AMR. At right we see this AMR formula identifier node replaced by a separately generated OPT with variable and operation type annotations (shown in the middle). Integrated formula+text AMRs were linearized as a token sequence and then converted to dense vectors using Sentence-BERT, and then used to re-rank Math Stack Exchange search results (Mansouri et al., 2022c; Reimers and Gurevych, 2019). One might instead use a graph neural network to embed from the graph structures directly.

Example 11: Augmenting AMR Trees with Operator Trees

Abstract Meaning Representation (AMR) tree, with inserted operator tree for formula in the query “Find $x^n + y^n + z^n$ general solution.”

---

2.4. **Indexing formulas and text**
In this chapter we have focused on representing formulas and text in an index. During indexing we create additional information sources, including dictionaries for source contents and formula annotations, and index files. As part of our discussion, we considered the information that formulas carry on their own, and in combination with surrounding text in a document. Related to this, we provided a brief overview of techniques used to annotate formulas with associated text for use in indexing. Symbol Layout Trees (SLTs) and Operator Trees (OPTs) are the most common formula representations, but other representations may be used such as visual-spatial representations like Pyramidal Histograms of Characters (PHOCs), and more might be developed. The integration of text and formulas in index representations is another place where research opportunities exist, perhaps including new ways to incorporate multiple formula representations and text annotations.
References


References


References


References


References


