The Structure of Mathematical Expressions

Thesis · August 2011

1 author:

Deyan Ginev
Jacobs University
30 PUBLICATIONS **122** CITATIONS

SEE PROFILE

JACOBS
UNIVERSITY

The Structure of Mathematical Expressions

by

Deyan Ginev

A THESIS FOR CONFERRAL OF A MASTER OF SCIENCE IN COMPUTER SCIENCE

Prof. Dr. Michael Kohlhase
(Jacobs University)

Magdalena Wolska
(Saarland University)

Date of Submission: August 31, 2011

Declaration

The research subsumed in this thesis has been conducted under the supervision of Prof. Dr. Michael Kohlhase from Jacobs University Bremen and Magdalena Wolska from Saarland University. All material presented in this Master Thesis is my own, unless specifically stated.

I, Deyan Ginev, hereby declare, that, to the best of my knowledge, the research presented in this Master Thesis contains original and independent results, and it has not been submitted elsewhere for the conferral of a degree.

Deyan Ginev
Bremen, August 31, 2011

Acknowledgements

I would like to first and foremost thank Prof. Dr. Michael Kohlhase for his continuous support and encouragement. This thesis would not have succeeded if he had not pushed me to try things out on my own, picking me up every every time I fell. I am also deeply indebted to Magdalena Wolska, who despite my stubborn Computer Science intuitions tried to educate me in the relevant parts of Computational Linguistics that I touch upon in this thesis, as well as to guide the treasure hunt for mathematical phenomena and their grammatical modeling.

The opportunities to work together with the amazing members of the KWARC group, to explore the borders between Computer Science, Mathematical Knowledge Management and Computational Linguistics and to have access to the rich large-scale document resources of the arXMLiv project, were definitely a great privilege.

I would also want to thank Bruce Miller for our insightful discussions that probed the differences between practical applications and theoretical modeling, as well us for being my mentor in understanding and adopting the LaTeXML software stack.

My gratitude goes to Pencho Yordanov and Stefania Dumbrava, for their critical comments and insightful suggestions.

"He who would learn to fly one day must first learn to stand and walk and run and climb and dance; one cannot fly into flying."

Friedrich Nietzsche

Abstract

This thesis investigates the representation of scientific knowledge in human knowledge bases, notably digital corpora of scientific publications, and works towards extending its capabilities to the technologies of the digital era. Concretely, the focus falls on the semiotic resource of mathematical symbolism and the process of making its semantics transparent to computational agents.

To set a solid footing, a systematic review of the structural phenomena in mathematical expressions is presented, aiming to uncover the variety of structural phenomena one might encounter. Throughout the thesis, a primary focus is placed on establishing the degree of influence that syntax plays in determining expression structure, also taking into account deep phenomena, such as ambiguity and underspecification.

A selected approach is to build a Combinatory Categorial Grammar, as part of a larger analysis pipeline, and to evaluate its recall, precision and certainty rates on a handpicked set of mathematical texts. We report a close to perfect recall rate with low degree of ambiguity, but lack conclusive data for achieved precision.

The second problem we address is to create a representation, and provide tools for its manipulation, adequate for the integration of logical parse forests into the state of art standards for content mathematics. This is achieved by adding minimal mixins to the existing content formats for mathematical objects, as well as by building generic notions for tree structure equivalence, compactness and sharing, in order to successfully capture underspecified and ambiguous expressions, contained as Compact Disjunctive Logical Forms (CDLF). This thesis provides rich support in terms of implementations of API access methods and auxiliary integration tools, in order to increase the usability of said enhancements.

To demonstrate the utility of the approach, we conclude with reviewing future work of possible applications of tools from the domain of Mathematics Retrieval, Semantic Publishing and Active Documents. While the phenomena overview and the introduced approach set a strong foundation for future work in the principle effort towards full disambiguation of mathematical expressions, enabling search on large-scale digital corpora, such as arXMLiv and Zentralblatt MATH, provides a decisive demonstration of the relevance of this problem and the high impact of our results.

Contents

Chapter 1

Introduction

This thesis investigates the automatic identification of structure of mathematical expressions in scientific documents. The provision of at least an "operator tree" for each expression is a minimal requirement to enable a range of shallow applications for scientific documents: semantic formula search, screen readers, interactive editing support and a clipboard for mathematical formulas, among others. More importantly, by constructing the logical skeletons of mathematical expressions, we build a foundation able to support the full formalization of said expressions. Synonymous to "computational understanding", this would open the door for deep semantic applications, such as definition lookup of objects, exploration of the theory graph of a document, in the context of a digital library of scientific knowledge, and ultimately – automatic verification of informal mathematical propositions.

In this chapter we introduce the societal context of mathematical expressions, follow their evolution into the digital era and discuss the motivation of directly exposing meaning to computational agents. We wrap up with an outline of the full thesis.

1.1 Evolution of Knowledge in Society

Human civilization has prospered and flourished due to the immense success of ever more complex societal systems. Societies have allowed for the separation and optimization of tasks, making possible the realization of goals far overreaching the potential of a single individual. We are not unique in our higher orders of organization, however. Most biological species form communities, which in turn integrate into ecosystems and thus comprise our biosphere. Remarkably, the fundamental systemic utility which enables functioning societies throughout the biosphere is, in its most abstract form, the same in all species. Namely, the ability of communication through species-specific "signs". The diversity of sign-systems occurring in nature is fascinating: ranging from chemical markers in ants, through sound signals in birds and higher mammals, to the gestures of chimpanzees. The sign systems employed become more elaborate and diverse as the species grow in complexity, culminating with the variety of signs employed in human societies. We will pay

specific attention to the sign system of human language, since it is the prime mechanism of constructing, representing and conveying human knowledge.

The development of our sign systems has been a vital corequisite in the evolution and growth of human societies. This process is captivating in its own right and has been studied in great depth in various scientific domains, notably History and Semiotics, the study of cultural sign processes. As we can not do justice to the complexity of such large-scale developments in human civilization, I offer but a brief and narrow overview. Initially, communication in human communities was oral and gestural, similarly to that of other higher mammals. Its purpose was procedural and present, motivated by *Material processes*, as dubbed by the systemic functional view of O'Halloran [O'H05, sec. 4.4, page 103], namely to aid and support the daily interactions in early societies. However, as interactions and communal systems grew more complex, so did the knowledge, or know-how, pertaining to their rules and functions. Consequently, trade was the chief culprit in motivating the invention of writing, as its procedural knowledge alone outgrew the capacities of human memory. Writing enabled the development of new sign systems, now unique to our own species – written languages. This provided the capacity not only to describe quantities for *Material processes*, but also to relate them and describe their properties, thus facilitating the evolution of *Operative processes* [O'H05, sec. 4.4, page 103].

The written sign systems guaranteed the long-term preservation of knowledge and paved the ground for the rise of reliable knowledge transfer and knowledge representation techniques. They also enabled a new type of communication process, scaling beyond the classic model of exchanging information between present participants – stable transfer of knowledge throughout space and time could now support targeting priorly unknown audiences, such as future descendants, or remote recipients. The peak of these novel opportunities can be seen today, where objective scientific knowledge is formulated and written in the form of modern mathematical discourse.

As this thesis is interested mostly in the transfer and utility of scientific knowledge, we narrow down our focus to scientific texts and their modern carriers. These texts embody various forms of mathematical discourse, which comprise a multi-modal, multi-semiotic resource, consistent with the principle decomposition presented at [Bat08]. Being an elaborate written representation of sign-systems, modern math discourse possesses three main semiotic resources as discussed in [O'H05, sec. 1.3, page 10]. We briefly outline their functional roles and some of their essential properties.

The "language" sign-system is informal, serving as a driver for the flow and understanding of a human reader. The "mathematical symbolism" sign-system is formal, contains condensed and unambiguously represented objective knowledge and is the main carrier of scientific truth. The "visual images" semiotic resource could be either informal, presenting a visual sketch of some entity, or formal, summarizing objective empirical results that are too numerous or complex and hence impossible to briefly represent symbolically.

1.2 Advent of the Digital Age

A system of such complexity did not emerge overnight. [Caj93] presents a detailed analysis of the gradual and continuous developments in mathematical discourse, which were needed for it to reach its present state. With the refinement and advance of scientific communities and their collective knowledge, so came the refinement and advance of the "mathematical symbolism" semiotic.

The 20^{th} century saw a rapid and decisive trend to evolve and standardize scientific notations and refactor inefficient practices by introducing global standards and modern typesetting paradigms. With the ever increasing participation of machines in the creation, distribution and archival of text, came a huge increase in the mechanization of human sign-systems and authoring methods. The adoption of computing technology caused a dramatic change not only in human communication, but also in human societies, enabled by the global network of the Internet. New capabilities were gained in all aspects of social interactions, notably in representation – video and sound could now be married to the simpler classical modalities, which in turn became active via hypertext and embedded computation. The digital age thus introduced an entirely new paradigm of possibilities for enhancing human communication and the general interaction with human knowledge, thus pushing societies towards an ever more scientific, globalized state.

It is important to remark that computers have only unified and greatly improved the functionality of the previous peaks in human communication – books for written language, telephones for speech and television sets for visual images, adding to them the power of an interactive, computational medium. In this sense, we have acquired a better utility for the distribution and preservation of signs, but have no right to claim a utility that operates with human knowledge itself.

And indeed, much effort has been invested into exposing knowledge to computational devices, in order to gain automated assistance with tasks such as verification, theorem proving, consistency and plausibility checking, reasoning, e-learning and search, to name a few. To this end, science has to meet the challenge of marrying sign systems to mental models for a new, symbolically computational kind of agent. Targeting one of the primary problems of the field of Artificial Intelligence, Mathematical Knowledge Management (MKM) investigates the properties, utility and representation of scientific knowledge. Similarly to the high impact improvements in the integration and utility of sign-systems that were enabled by the (X)HTML formats, for what are now known as "web documents", MKM attempts to pave the way to exposing the semantics of those signs, i.e. the knowledge itself, to the mechanical agent.

It was only recently that all semiotic resources related to mathematical discourse have been made accessible to the computer in a machine-comprehensible manner. Current efforts to the creation of open representation standards provide presentation of the semiotic signs in a machine-readable state, i.e. via Presentation MathML [ABC+10] for "math symbolism" and SVG [JFF02] for "visual images". The "language" resource is represented in (X)HTML [The02, RHJ98]. CSS [Bos98] and embedded computation (e.g. JavaScript) can further augment any of the resources, providing dynamical behaviour and decoupling sign typography from the sign containers themselves.

Additionally, computer-oriented formats that expose the discourse semantics have been designed for the symbolic resource (Content MathML [ABC+10], OpenMath [BCC+04]), while a generic approach to exposing a subset of the discourse semantics is being developed for all signs serialized as XML, via the RDFa standard [ABMH]. Holistic approaches to concrete ontological domains exist as well, one of them being OMDoc [Koh10]. OMDoc acts as a markup format and data model for mathematical knowledge, attempting to capture all of the relevant semantics, be they inter- or intra-semiotic, uni or multi-modal, explicitly structural or implicitly contextual. Hence, it incorporates (subsets of) several of the more fine-grained standards discussed above, notably OpenMath/Content MathML and XHTML (from OMDoc 1.3).

1.3 Making the Next Step

The big problem with exposing the semantics of human discourse to machines, is caused by the syntactic nature of semiotic signs, as well as the formal symbolic essence of computing, as opposed to the semantic nature of human knowledge, available only via a human interpretation of a given discourse representation.

This thesis attempts to make a fundamental step towards the explicating of semantics, by focusing on mathematical symbolism and more precisely, mathematical expressions.

As the symbolic resource is formal, it is an immediate first candidate for exposing to computational agents. To reiterate, human knowledge has been encoded in sign-systems that convey a syntactic presentation, while the semantics have been in the interpretation of the reader. This is also true in the case of math expressions, although the interpretation is more direct than in natural language, since the symbolic resource was created so that "meaning is *unambiguously* encoded in ways which involve *maximal economy* and *condensation*" [O'H05, sec. 4.2, page 97][1] and "to describe patterns which can be rearranged for the solution to problems" [O'H05, sec. 4.6, page 118]. These principles are central to computational processes themselves, thus alleviating the problem of the interpretation of signs, to the extent where one needs to solely determine the scope and implications of an expression's context.

Furthermore, much of the existing scientific knowledge has now been digitalized, most prominently large-scale archives of scientific publications (arXiv [ArX], Zentralblatt MATH [ZBM]), while new knowledge bases are being created in ways explicitly exploiting the novel computational and societal capabilities (Wikipedia [Wik07], WolframMathWorld [Mat], PlanetMath [Pla], among others). Efforts such as the arXMLiv project [SKG+10] are starting to mature, so that these archives could now be represented in a transparent form to automata. The state-of-craft in services dealing with digital documents is still largely limited to presentation, but early approaches to content-driven services have also been undertaken. Perhaps most notably, the Planetary system [DGKC10] has recently been developed to combine the social and semantic computational capabilities of the digital era, setting the stage for the first "social-semantic services" for scientific documents. However,

[1] Note, however, that O'Halloran referred to an unambiguous process for human readers fully aware of both context and a sufficient extent of background knowledge.

such emergent systems have their sight set strictly into the future, where they can fully control and prescribe the techniques and languages an author should use.

The collective volume of scientific knowledge already compiled throughout history far exceeds the capacities of an individual's memory. What's more, it is clearly impossible to even familiarize one's self with a small fraction of that collection in a single lifetime. Clearly, we should not ignore a wealth of knowledge so daunting in its vastness by only focusing on authoring tools fixated at the future; this thesis aims at setting the stage for transforming an important subset of the existing scientific knowledge into a content form. Also, it demonstrates some of the first benefits from introducing a computer companion in the process of understanding and examining the signs of written language, particularly of written mathematical symbolism.

1.4 From Presentation to Content – an Overview

The intended central contribution of this thesis is to present a solution to the extraction of content trees from the signs of mathematical symbolism, concretely from their digitally encoded XML representations. In the field of Math Recognition (MR), such process is divided into two parts – the reconstruction of syntactic structure (creation of "layout trees") – and the consequent extraction of a content representation (creation of "operator trees"). We provide a fully detailed overview of the state of art in Chapter 2. Our focus would be the process of meaningfully and exhaustively constructing operator trees from layout trees, since we target XML-based digital corpora of mathematical documents, which already contain a representation of the layout trees.

We investigate this problem in two independent stages. The first is the process of extracting meaning from presentation, where one needs to address and accommodate the various types of linguistic phenomena which arise. We address these issues in detail in Chapters 3 and 4 and discuss our approach in Chapter 5. Chapters 6 and 7 detail the process of development and evaluation of our method, reporting on different measurements of its performance. In Chapter 8, we retract our steps to discuss the relevant related work in the field, from the new perspective of having evaluated our achievements.

The second, follow-up challenge, is to design a representation for the extracted meaning. We discuss in Chapter 9 that existing formats do not support natively semantic expressions that are underspecified or ambiguous. Additionally, in order to ensure the efficiency of further processing with the representation, as well as to optimize it in terms of size and semantic clarity, one also needs to address the challenges of representation redundancy and cross-referencing. To this extent, this work provides a minimal set of conservative extensions, as well as respective thorough API support.

In Chapter 10, we conclude the thesis and additionally attempt to evaluate the utility of our extracted underspecified semantics, by prescribing the software stack one would need in order to index and search the digital corpora we have access to with the MathWebSearch [KŞ06, PK11] search engine, addressed in detail in Section 10.1. We also discuss a range of high-profile applications that could be directly enabled by building on the results described above.

Chapter 2

State Of Art

In Chapter 1 we examined how the overall importance of the problem of constructing content representations of mathematical expressions can be best understood in the full context that relates the fields of Semiotics, Artificial Intelligence, Mathematical Knowledge Management and Computational Linguistics. The processes in question are of conveying, representing and deriving human knowledge via semiotic signs, specifically those of the "mathematical symbolism" semiotic resource. We are interested in constructing a solution for recovering the structure, as a first step to recovering the full meaning, of mathematical expressions. Any solution should transform signs into logical forms in a content representation, to make them directly available to semantic applications.

It is fascinating that this problem has been approached from a range of perspectives in different scientific domains, with a variety of foci. Below we survey the state of art, paying close attention to the extraction of structure in mathematical expressions, the representations of the logical forms, as well as the utility and technical trade-offs of the different approaches.

2.1 Work in Language Understanding

While the linguistic analysis of scientific documents is widely regarded as an interesting line of research, the actual body of work in the field is still rather limited. This is not a surprise, as mathematical documents form a rather specific linguistic niche that poses a range of special domain challenges. We could contrast it to the domain of bio-medical documents, which has been subject to thorough investigation by a large number of research groups in recent years.

Two of the current main challenges that make mathematical documents challenging to work with are: adopting and working with a complex representation capable of hosting natural language together with mathematical symbolism; the absence of well-prepared annotated linguistic corpora. And indeed, the state of the art tries to largely circumvent these problems by restricting the scope of analysis to well-behaved controlled discourse

fragments, working mainly "in the small".

We will examine two directions of research that have shown good progress in recent years.

2.1.1 Controlled Natural Language for Mathematics

One approach is to incrementally build a "controlled natural language" for mathematics, capable of supporting a sufficient subset of natural language that would allow an author to write mathematics in an intuitive manner, but also be restricted enough to allow an unambiguous formalization of the input. As this approach has the primary goal of building formalized libraries of mathematics, it focuses on establishing the full analysis and formalization pipeline over a narrow coverage of language, followed by a step by step expansion in coverage.

Active projects in this field are FMathL [NS10], MathLang [KMW04], MathNat [HR10] and Naproche [CKS11].

As the input language is controlled, and the domains of experimentation are still narrow (typically arithmetic, set theory and calculus) most approaches make due with context-free fragments of mathematical expressions. However, it is a common intention to extend beyond those fragments, which motivates choices of advanced parsing tools. MathNat bases its syntactic parsing on the Grammatical Framework (GF) [Ran04], a typed functional language based on a type theory inherited from logical frameworks. FMathL also takes advantage of GF, but is currently also developing its own specialized parser for mathematical discourse [KN11], aiming at efficient parsing with dynamic rule addition, with the expressiveness of parallel multiple context-free grammars (PMCFGs). MathLang does not commit to a specific parsing framework and instead has an open-ended array of "aspects" that can introduce various analysis techniques. The "Core Grammatical aspect" uses a custom weak type system and reasoning techniques to parse mathematical discourse in context.

The Naproche system is based on a modification of Simple Type Theory (STT), following Ganesalingam's intuitions [Gan09] for an adaptive understanding of mathematical language. Notably, the process of extracting structure from mathematical expressions is explicitly handled in a context-free "Normal Formula Grammar" and described in depth in [CKS11].

As a result of pursuing a complete processing pipeline, none of the current efforts has introduced a transparent representation to their analysis results, e.g. using open XML standards such as OpenMath [BCC$^+$04] or OMDoc [Koh10]. Instead, the results are passed as objects in the data structures required by the tools at the respective processing stages. All subsequent applications that have been investigated in these projects are fully formal: automatic verification, theorem proving, as well as integration with computer algebra systems.

Among these projects, there is universal agreement on the need to (ultimately) account for and resolve ambiguity and underspecification. The methods for such resolution vary, but invariably utilize context in order to change a dynamic state of a parser, or infer type

information in order to reach an unambiguous derivation in a type system. As these efforts are within a setting of a controlled input language, there is an ultimate requirement of reaching an unambiguous representation of the modeled expressions and sentences, either by reasoning within the systems or by prompting the author for further definitions on input.

2.1.2 Natural Mathematical Discourse

The alternative to the controlled bottom-up approach, is to explore and model the original language of scientific documents. It is common to investigate the phenomena in a restricted domain or type of discourse, for example the Dialogue project [BB05] investigates "*flexible natural language tutorial dialogue on mathematical proofs*". Ganesalingam's "The Language of Mathematics" [Gan09] focuses on foundational questions in a set theoretic setting, while Zinn [Zin03] looks into "*short and simple proofs that were taken from textbooks on elementary number theory*".

As in the controlled approach to language understanding, a full pipeline down to formal understanding is typically pursued. Considering the embedding of symbolic expressions in mathematical discourse, it is easy to understand the motivation and immediate benefit of having a formal representation of at least such "formal-near" citizens. However, this requirement is typically relaxed, both because it is hard to achieve large coverage on unconstrained language and because there are semi-formal and informal fragments interlieved in the discourse. As our current interest is to reveal the structure of mathematical expressions, and not yet their unambiguous semantics, we will not go in depth in reviewing the semantics construction and semantic analysis stages of these projects. We only briefly remark that language underspecification and ambiguity are given explicit treatment by using Discourse Representation Theory [Kam95] as the semantic representation, except in the case of Ganesalingam, who takes the stance that all prior information needs to be explicitly provided in the discourse, positioning himself on the border between controlled natural language and full discourse.

Consistent work in the field has been carried out by Magdalena Wolska, both in a vertical direction towards formal representations and deep semantics [WKK04, HW06b], as well as in a horizontal, knowledge-poor view on mathematical discourse [GWK09, WG10, WGK11]. The corpus used for the latter shallow experiments is the arXMLiv [SKG+10] archive of scientific documents. In this thesis, we will also use a document sandbox from arXMLiv to base the development of a grammar for parsing mathematical expressions.

The parsing of mathematical expressions, where applicable in the projects referenced, is knowledge-rich – again motivated by the desire to have a full formalization of the discourse as a final outcome. It is performed by a combination of a context-free grammar and a type system, or a grammar directly based on a type theory, such as GF or Combinatory Categorial Grammar(CCG). Since the current work takes a knowledge-poor view on the discourse, we will observe in Chapters 4 and 5 that while parts of the methods apply, we will have a different set of considerations.

Nevertheless, the representation question tends to be circumvented via normalization

techniques or focusing on simpler structures, as indicated by the symbolically simple domains. To contrast, quantum physics or differential and algebraic geometry exhibit a much richer pool of symbolic constructs, as we discuss in Chapter 7. What these projects do achieve is to unearth a rich spectrum of linguistic phenomena that occur in mathematical discourse, as well as a range of techniques for accommodating them, on which this thesis builds on.

2.2 Work in Mathematical Knowledge Management

If we do not start off with a mindset of wanting to understand mathematical language, but have a primary purpose of managing collections of mathematical discourse instead, we emerge with a very different set of intuitions. MKM seeks the creation of accessible digital representations of accessible knowledge, in order to facilitate building digital libraries of scientific documents and to enable a wide range of user-assistance document-embedded services. It hence also aims to provide the tools to transform legacy carriers into said representations and to ultimately exploit the document semantics for services that integrate systems for theorem proving, verification and computer algebra.

We will review the relevant work in the field grouped in three subcategories, corresponding to three subdomains of MKM research.

2.2.1 Mathematics Recognition and Retrieval

For a detailed overview over the fields of math recognition and retrieval we refer the reader to Zanibbi and Blostein's review "Recognition and Retrieval of Mathematics" [ZB11]. It presents the only holistic analysis of the processes of extracting, representing and utilizing mathematical expressions, of which the author is aware of.

The first challenge of recognition techniques is the recovery of a "layout tree", which is sufficient to provide an editable digital document to a user, as well as to create modern presentational versions of the document, such as (X)HTML with Presentation MathML. The next step is creating "operator trees", namely data structures that represent the logical relationships within an expression, as opposed to its presentational horizontal and vertical relationships. Such trees would make computationally transparent the structure of the mathematical expression, which is a necessity for semantic applications such as text-to-speech of mathematical expressions, semantic (or content) formula search, etc. Still, Zanibbi and Blostein, whose work is in math recognition, largely overlook the challenges specific to operator trees and tend to underestimate the challenges as solvable by heuristic methods. We must also remark on the additional burden of the uncertainty in the correctness of the layout tree, which adds to the difficulty of establishing the expression's logical structure.

2.2.2 Digital Mathematical Libraries

Apart from digitalizing printed texts via the methods of mathematics recognition, digital libraries are also constructed by transforming retro-born-digital documents, originally written in typesetting systems such as TEX/LATEX, into a MKM-compliant XML representation. Two large-scale examples are arXMLiv [SKG+10] and EuDML [EuD], that use the LaTeXML [Mil] and Tralics [Tra] converters, respectively.

While Tralics makes no attempt to recover the logical structure of mathematical expressions, LaTeXML does. For more details, [SGD+09] offers a comparison of the different conversion softwares currently available. LaTeXML uses a context-free grammar to heuristically establish the logical structure and supports post-processing exports to Content MathML and OpenMath. We will provide a more in-depth look into the LaTeXML grammar in Section 8.1.

The challenges come from being in an almost completely knowledge-poor setting, with the small exceptions of semantic cues provided via the regular LATEX markup. However, that markup can be abused for presentational reasons and is also not inherently reliable. In that sense, there is relatively little one could do without inferring additional semantics on all document levels. This thesis seeks to explore exactly this space of knowledge-poor corpora, attempting to establish the reliable syntactic markers that can be used in arbitrary scientific domains.

2.2.2.1 Authoring with Semantic Markup and Semantic Publishing

As the digitalization of existing documents poses its own set of limitations, there are concurrent efforts to provide authors with the tools for creating documents compliant with the MKM vision of a digital library that exposes as much of the semantics as possible, to the machine. Two of the prominent efforts in this direction are STEX [Koh04] and PLATΩ [WAB06], each of which provides an extension to LATEX that allows authors to explicitly embed the intended semantics of discourse theories, statements and objects, as well as document metadata. Both projects generate a content representation of the document in OMDoc, which they use as an interface language – STEX for the Planetary semantic publishing platform [DGKC10], and PLATΩ for the ΩMEGA proof assistant [KMS99].

In terms of extracting the logical structure of the embedded formulas, PLATΩ expects an already formalized input in its own controlled proof language, while STEX allows for flexible macro definition of explicit markup for natural language expressions. From this perspective, PLATΩ is probably closer to a controlled natural language, while STEX is a flexible framework for semantic markup. There is great potential value in reusing document sets encoded with such tools, as they can be turned into a gold standard for training linguistic tools relatively simply. In the case of STEX one could use the OMDoc representation as the annotated representation, and have a parallel conversion to the regular XML format of the arXMLiv project as the original source to be subjected to analysis.

2.2.3 Interfaces for Automated Theorem Provers

An interesting and not immediately obvious source of insight is rooted within the efforts of designing input and output interfaces for Automated Theorem Provers (ATP). As the ATP developers strive to increase the usability of their tools to their users, it has become apparent that a more relaxed language for specifying the input is necessary, as well as a more flexible and customizable way of rendering the results.

On the input side, the MOWGLI project [CZ04] allows for ambiguous declarations, focusing on resolving the exact typing information. As the languages of theorem provers are either completely or close to fully formal, the emphasis falls on disambiguating the correct types of the expressions, which would induce an overall unambiguous semantic expression tree. What they do not cover in depth, and do not need to address, are cases where the underspecification does not allow for successful type recovery from the start, but only when the content tree of a formula has been built, or not at all. While the latter is the problem we address in this thesis, [CZ04] provides an excellent reference point for attempting type recovery, especially when more domain knowledge is present.

As for the output rendering, there are attempts to serve the results in the notations to which each user is accustomed to. Libbrecht's notation census [Lib10] and Smirnova's notation selection tool [SW06], provide an excellent window into the relation between mathematical notations and their underlying meaning. When used in reverse, they reveal important phenomena regarding the use of various mathematical symbols, which we discuss in Chapter 4.

2.3 Consolidating the Disciplines

We could generally state that the extraction of logical forms from mathematical expressions has not yet been centrally addressed so far. There exist well-motivated initial efforts on small collections of expressions, as well as heuristic solutions in the large. However, a systematic analysis of the syntax of mathematical expression is lacking, as well as a clear separation between syntactic and semantic phenomena – probably a side-effect of the practice of building full pipelines of analysis that closely couple the two.

In attempt to address the representation challenges and to offer a ready setup for large-scale analysis, the author co-founded an effort towards a framework for Language and Mathematics Processing and Understanding (LaMaPUn) [GJA+09], which is now reaching a stage of maturity. The purpose of the LaMaPUn framework is to fully encapsulate the knowledge representation part of the challenge into its internal abstraction layer and to provide an API, backed up by an Ontology, for standardized access and querying of the resulting knowledge base. The framework stays close to the technologies employed in active mathematical documents, rooting itself on the Semantic Web technology stack [AvH04] and hence remains intuitive for further development, via its distributed module system for linguistic analysis. In following these or similar principles, the author is not aware for other related work in this direction.

Nevertheless, there exist tools that take a side entrance to obtaining document semantics (e.g. via semantic markup), which are starting to be adopted. Applications based on semantic documents have existed for some time now, and are still being actively developed – in Information Retrieval [AS04], Computer Algebra Systems (CAS) [DLR10], Theorem Proving [CZ04, BMS03], Semantic Publishing [DGKC10] and Active Documents [DLR10], among others. In this regard, the serialization of the logical forms of mathematical expressions has been largely realized via the OpenMath standard [BCC$^+$04], while the format of choice for the creation of web documents has been Content MathML [ABC$^+$10]. As of Content MathML 3, the two formats have now conveniently converged.

To conclude, using the state of art as a reference point and motivated with applicable use cases, we proceed to examine the phenomena of mathematical symbolism in the following chapters, making an attempt to systematize and accommodate them accordingly.

Chapter 3

Ambiguity and Underspecification in Mathematical Expressions

The central cause of complexity in making mathematical knowledge transparent to automata is the presentational nature of its written semiotic signs, which induces various forms of ambiguity. Additionally, these signs are always presented in a context assumed to be implicitly clear to a reader, who is also expected to possess a certain amount of background knowledge. Since computational processes are currently very far from being able to implement a human theory of mind, these challenges are still open and in need of solutions.

3.1 A Brief Examination of Mathematical Symbolism

Contrary to natural language, the symbolism of mathematical expressions has a rather small fixed lexicon in regular use, notably the letters of the English and Greek alphabets, as well as a variety of special mathematical tokens ($+, \sqrt{}, \langle\cdot\rangle, \ldots$). The range of mathematical symbols is presented in a variety of resources. The MathML specification [ABC+10] has about 2500 symbols[1], while "The Comprehensive LaTeX Symbol List" [Pak09] contains 5000 LaTeX symbol-forming command sequences, many of which, however have overlapping semantics, and only offer different styling or presentation.[2]

From a historic perspective, it is interesting to observe that the spread of newly created symbols has decreased in the last century, to the extent that mathematical symbolism has reached a very stable state in the range of symbols employed across all disciplines. The author suggests that the digitalization of scientific documents is at least partially responsible, as it has made custom notations harder to introduce, since they require a higher investment on an author's behalf. In computerized typesetting systems, adding a

[1] see `http://www.w3.org/TR/xml-entity-names/`

[2] This is not to state that all such symbols are equivalent, as is clearly the case with N and $\mathbb{N}$, which have different semantics even though they are but different fonts for the same letter.

new token typically requires creating a custom glyph, possibly in a new font, and convincing others to adopt it, which is a lot more work than just drawing a new symbol on paper.

However, the mathematical vocabulary remains open-ended, via the use of well-defined mechanisms for the creation of complex tokens. Objects such as $C_{\Delta^n,f}$, permit two-dimensional stacking of symbols, in order to compose subexpressions into a larger entity. These subexpressions, as well as their mutual relationships, have a wide variety of possible functional roles, which we will not cover in depth here. We should, however, make a remark, that the reinterpretation of symbols is openly permitted, e.g. when building up the vocabulary of a novel subdomain of Mathematics, and so is the reinterpretation of structural interrelations between an expression's tokens.

Also differently from language, mathematical objects have well-defined and more evident scopes and contexts, e.g. via explicitly labeled discourse structures, such as "Definition" paragraphs. This is necessary, as mathematical texts need to be flexible with the semantic interpretations behind a given sign, in order to succeed in developing new notations while keeping the allowed lexicon small and fairly standardized across disciplines and communities. While a typical natural language lexeme would have a rather fixed set of literal meanings, the signs of mathematical symbolism have the potential to be redefined in each text and sometimes carry a variety of independent semantics across the same discourse. One example would be a typical use of complex signs having a head "P" in the mathematical domain of Probability Theory. We see them employed with the semantics of a probability distribution(P_*), but also of a hyper probability distribution(P_{**}) which is one order higher, in the same expression, namely the Bayes' formula[3]:

$$P(\Theta \mid D, M) = \frac{P_*(D \mid \Theta, M) P_{**}(\Theta \mid M)}{P(D \mid M)}$$

Modern approaches in Computational Linguistics generally split into two scenarios, differing by the built-in capabilities of the analysis procedures and sharing a complementary set of pros and cons.

3.2 Knowledge-poor Settings and Underspecification

In corpus-driven settings, the focus falls on efficiently analyzing a large-scale set of documents, usually by employing shallow statistical techniques. As discussed in Section 1.3, such corpora of scientific mathematical documents exist and are finally becoming accessible for computational linguistic analysis.

A standard setup in such scenarios is a data-rich, knowledge-poor approach, where the analysis limits itself to statistical methods and shallow techniques for recovering intra-document context and syntactic structure. An example is [Gri10], where a knowledge-poor automated method for disambiguating symbolic expressions in mathematical texts is developed, based on the arXMLiv corpus [SKG+10].

[3]The asterisks are added for labeling clarity and are not part of the original Bayes formula.

Knowledge-poor approaches fine-tune the trade-off between efficiency and complexity of the analysis approach, by abstaining from using any prior contextual or domain knowledge. All conclusions are derived (usually in the form of statistical correlations) from the vast corpus resources.

In the concrete case of mathematical symbolism, where mathematical tokens and objects posses a very wide spectrum of semantic interpretations, this induces a vertically deep underspecification of the expression components. Such underspecification could partially be resolved by contextual analysis, which could hint to the correct domain of the article. This is a very difficult process, however. The "Mathematical Subject Classification" [Soc09], an effort to build a taxonomy of mathematical disciplines, already contains close to one hundred domains on its top tier, which in turn have a variety of partially overlapping notational conventions and symbol semantics. This clearly places underspecification as one of the primary challenges for knowledge-poor approaches.

3.3 Knowledge-rich Settings and Ambiguity

In contrast, some settings provide additional knowledge, or "metadata", about the discourse being analyzed. Most modern corpora (e.g. arXMLiv, Zentralblatt MATH, PlanetMath) provide a classification indicating the domain of the document, as well as external information about the submission procedure or authoring process. The presence of at least basic, but preferably rich, metadata is a preliminary condition for the development and utility of knowledge-rich analysis approaches.

Knowledge-rich approaches augment the agent with additional reasoning techniques and specific judgements, based on a priorly accumulated knowledge base and what typically would be considered "implicit context" awareness. For example, knowing that a document is in the domain of Newtonian mechanics, as well as the signs of the units and physical constants conventionally used in that domain.

In such a setup, underspecification is resolved by domain knowledge, providing clarity essentially for named constants and objects in the domain. When addressing the syntactic behaviour and meaning of variables, operators and structural relationships, however, one easily discovers that multiple issues of ambiguity need to be addressed.

An example, introduced in [CZ04], is the syntactic parse of "$a = b \wedge c = d$", in the domain of formal logic, which depends on the types of the objects a,b,c and d. Essentially, when they represent terms, the only disambiguation is to interpret $=$ as term equality, hence building the parse corresponding to "$(a = b) \wedge (c = d)$". If the tokens are first-order formulas, however, the syntax might also be the one of "$a = (b \wedge c) = d$". Numerous sibling ambiguous phenomena exist, based not only on token interpretation, but also on unclear symbol roles (operator, delimiter, object) as well as structural syntax – operator scope, precedence and dominance. More convoluted ambiguities include "invisible" participants in mathematical expression, for example invisible operators (invisible times, invisible plus, function application), as well as argument suppression, e.g. skipping the ranges of a summation symbol ($\sum$).

As ambiguity is rooted in the multiple possibilities of creating syntactic or semantic parses of a given expression, a natural approach to resolve it is by employing intra- and inter-expression context. The greater the understanding of the context, the better disambiguation could be accomplished. A particular instance of this principle is that greater size of an expression, usually corresponds to a much lesser degree of semantic ambiguity, as it provides greater intra-expression context. A systematic document-level analysis, would in turn provide greater inter-expression context, thus solving harder cases such as correctly scoping symbol definitions bound locally to a particular theory or definition. This motivates solution approaches which rely on the maintenance and evolution of a dynamic state, e.g. dynamic grammars.

In the course of this thesis, we focus on developing knowledge-poor solutions, but will also briefly address some examples of implementing knowledge-rich techniques.

Chapter 4

Structural Phenomena in Mathematical Expressions

In this chapter we present a systematic account of the structural phenomena in mathematical symbolism, in an incremental fashion. Our main goal is to establish domain-independent phenomena that ultimately contribute to driving a parsing process for generating the logical forms of the input expressions. However, we will also pay attention to phenomena induced by domain-specific uses of lexemes, such as polysemy, as their diversity induces the degree of underspecification of an expression's logical form.

There is an important and counter-intuitive prerequisite to perform such an investigation, namely to initially treat mathematical expressions as completely syntactic, establishing the patterns and conventions of the compositional interplay of their atomic signs (or lexemes). Once such a knowledge-poor basis is established, we also discuss the various complex constructs in mathematical discourse (such as transfix operators), which we attribute to domain-specific conventions.

Scope of analysis The author's observations have been performed on modern western mathematical notation in scientific publications, authored in the last decade of the twentieth century or later. We contrast this setup with the notations of Arabic mathematics, which have a reverse expression flow and are read from right to left, as well as with early notations which were typically less standardized and more graphic (see [Caj93] for a detailed overview).

Another restriction is to focus on a single modality of mathematical symbolism, namely that of "inline expressions", interleaved in natural language sentences. We make this restriction to avoid various modality-specific phenomena that we are not interested in at the very beginning, such as multi-line expressions, equation groups and complex two-dimensional structures (e.g. matrices, piecewise functions). Lastly, we will consider only formal expression fragments, which do not have interleaved natural language within the expression body. An example of an informal expression that we consider out of our scope is $\{x \mid x \text{ is prime}\}$.

Concretely, we performed our study on a range of resources described in Section 6.1. As our primary goal was to spot cross-domain invariants, the author compiled these resources in order to ensure sufficient diversity, but also complexity and representativeness of the texts. We have reviewed all expressions in our discourse collection, and report of our findings in full.

Modern day science, developed by a truly global community, is in a state that has sincerely embraced standardization and digital authoring tools. The language of mathematical symbolism is celebrated as one of the truly universal languages[1], together with musical notation, which further adds to our motivation to analyze its cross-domain syntactic properties.

4.1 A Hierarchical View of Mathematical Symbolism

In order to obtain a fine-grained understanding of the causes and occurrences of the different syntactic phenomena in mathematical expressions, we build a layered view on the syntax of mathematical expressions. The purpose of this gradual increase of assumed lexical knowledge will become apparent as we demonstrate the phenomena of each level with illustrative examples.

4.1.1 Knowledge-poor extreme

Ganesalingam notes that a mathematical symbol could be seen as a black box with decorations:  [Gan09, "diagram of positional operations in math symbolism", page 36].

Thus, we can define an inline mathematical expression as a sequence of such symbols:

 (4.1)

4.1.2 Scripts

Since we are interested in obtaining the operator trees, let us make the following assumption. The author conjectures that, syntactically, scripts act as modifiers for their baseline carriers and never influence the rest of the operator tree.

This seems to be the case regardless of the syntactic role of the base lexeme (e.g. object, operator, fence, delimiter). Note however, that the induced semantics might be much more complex:

1. to form complex names: $\tilde{x}$, G_{Δ}
2. to modify (type, restrict range): typed equality $=_{\alpha\to\alpha\to o}$

[1]Note that the universality of mathematics is mostly in structure and the core lexicon. Indeed, many idioms and custom notations exist which diverge from the each other.

3. to denote an operation: inverse of a function f^{-1}, powers x^n, derivatives f'', range selectors $\mathbb{N}_{>2}$

4. to act as arguments: ranges of sums and products $\sum_{i=0}^{\infty}$

Another problem arises in situations with multi-scripted symbols, when one needs to decide on the order of the scripted actions. A revealing example is taking a vector x and writing x_1^2 – does the author intend to square the first element of x: $(x_1)^2$, or take the first element of the squared vector: $(x^2)_1$? These considerations can only be made in a setting with at least some domain and type information and are hence out of the scope of the current study. If the scripts semantics are left underspecified, the order of the scripts does not carry any meaning in of itself, which would postpone the issue to a later stage of understanding.

Consequently, we will look into inline expressions, while disregarding their scripts. We will establish that this does not hinder us from understanding the structural phenomena occurring on the baseline.

Hence we reference (formal) inline expressions to be of the form:

$$\cdot\ \blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare\cdot\blacksquare \tag{4.2}$$

We adopt this as a running example, corresponding to the mathematical formula:

$$a\left[b + f\left(b, b\right)\right] = 100 \tag{4.3}$$

4.1.3 Concatenation operators

When underspecified, as in Eq. 4.2, we can treat each ■ symbol as connected to its direct neighbours by an invisible concatenation operator ($\cdot$ in Eq. 4.2). We justify this assumption as follows.

Both the syntactic function and the semantics of ■ are completely unknown, hence it could be syntactically seen as "anything". It is a well-known phenomenon of basic arithmetic to omit implicit operations when they are obvious from the context, for example $2a$ or $2\frac{1}{2}$ have two adjacent symbols that are syntactic atoms[2], but form grammatical expressions with unambiguous semantics, namely those of "2 times the number a" and "2 and a half" (or "2 plus one half"). In these examples, the presence of a concatenation operator is evident, as well as its semantic contribution. In the author's experience, there are at least five possible distinct semantics of a concatenation operator:

1. Invisible plus: $2\frac{1}{2}$

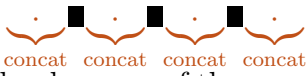
2. Invisible times: $2a$

3. Function application: $f(a)$

[2]We treat $\frac{1}{2}$ as an atom, which we clarify in Section 4.1.6.5

4. Constructor of geometric objects: a segment AB, a triangle, or an angle, ABC

5. Invisible delimiter:[3] permutations $(a\,a\,b), (a\,b\,a), (b\,a\,a)$

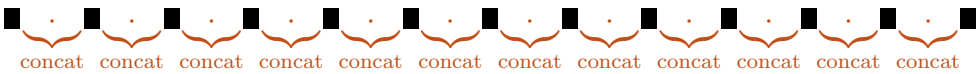
Having demonstrated the existence of concatenation operators with their own semantics, we turn to the other cases. Let us take a basic example: $1+2$, with the underspecified form 

Clearly, none of the concatenation operators carry semantics in this case, as the expression denotes an application of arithmetic addition to one and two. There are two points of view on this phenomenon. We could either postulate that the concatenation operators simply do not exist or that they are void of semantics and act purely as syntactic glue. It is hard to see which interpretation is better, as there is simultaneously no lexical evidence of the presence of these invisible operators, but also the need of a uniform syntactic treatment of concatenation. In other words, we can either state that concatenation exists only when it has a semantic contribution, which could be hard to determine in an underspecified setting, or that it is uniformly present but not always semantic, which would in turn lead to an increase in structural ambiguity.

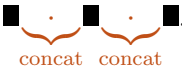
The author believes that both views carry their value. In this exploration, we start with ubiquitously depositing concatenation operators around each symbol and we shall gradually discard those that are clearly non-semantic, based on the surrounding intra-expression context. We mark such observations with **Concatenation Rule** as we go along.

Concatenation Rule 1 Concatenation operators should only act as connectors between ■ symbols, as they never carry semantics in the start or end of an expression. Hence, we shall discard any leading or trailing concatenations of an expression.

Continuing with the running example we have introduced in Eq. 4.3, we reach the expression:



(4.4)

I decide to preserve the occurrences of concatenation which could be potentially void of semantics, when this can not be unambiguously decided from the syntax only. As an example, let us take aRa, with an underspecified form 

. On one hand, both concatenations could have the semantics of invisible times, or alternatively, they could be void of meaning and act as syntactic glue, e.g. if this was an example of reflexivity

[3] One might argue that a good style for invisible delimiters would require an amount of whitespace which should be treated as a semantic operator, instead of the concatenation operator. The two views are probably equivalent.

where the infix relation R is applied to two arguments a on both its sides. Interestingly, these considerations apply to infix relations, but not to higher-order prefix relations such as $Rxyz$ (the relation R applied to three arguments x,y and z).

It is probably indicative that a human reader would not notice concatenation operators when they do not contribute to the semantics of the expression, which is the intuition we are trying to model towards. If we look into classical simple function application, such as Fx or $f(x)$, and cascading syntax for relations such as $Rxyz$, we see that we have the same intuition of what the concatenation operators mean, namely function application. If we generally compare Cartesian syntax of function application, such as $f(x, y, z)$ with the cascading alternative $f\,x\,y\,z$ we can distinguish a function symbol f being applied to an argument sequence/vector/list (x, y, z) or a higher-order function f which participates in three subsequent applications. The two syntaxes are only syntactic sugar for each other and are semantically equivalent, but they nicely reveal how the intuition for applying a prefix function (or relation) translates into an infix concatenation operator with "function application" semantics.

We draw the distinction between the semantic concatenation in $Rxyz$ and the non-semantic glue in aRa on a purely syntactic basis – in the latter infix case, a human reader has the intuition of R being applied to two arguments, which can not be compositionally represented via concatenation operators in an intuitive manner. An attempt to do so results in a counter-intuitive interpretation of "apply-left" and "apply-right", which is not a convention that reveals serious benefits and would instead artificially attribute application semantics to a syntactic attachment. What we adopt instead is a convention that a concatenation operator can act as prefix and suffix (or postfix) "glue", that will be void of semantics. It accommodates the infix application of R, when seen as an infix relation. This treatment also allows the desired ambiguity of the various semantic interpretations of concatenation to remain grammatical.

4.1.4 Fences

Having discussed all attachments to the baseline ■ symbols, we turn to reviewing and categorizing the syntactic behaviours of the various ■ lexemes. To start, we consider what seem to be the most universally adopted symbolic lexemes – expression fences.

With "fences" we understand pairs of left and right fence symbols that act as grouping circumfix operators, isolating a subexpression from its surroundings. Exhibiting a wide range of semantics, the syntactic role of fences is to group subexpressions, explicitly marking stand-alone subtrees that should be approached individually and that should remain clearly separated from the rest of the expression.

The well-known and universally accepted mathematical fences are parentheses $(\cdot)$, square brackets $[\cdot]$ and braces $\{\cdot\}$.[4] This statement is also confirmed by the Presenta-

[4]The author is aware of a trend in digitally authored mathematics, to move away from brackets and braces as grouping operators and universally adopt parentheses instead. This is probably induced by the introduction of automatic mechanisms for stretching fences, making them visually appealing without any added authoring cost. The need of clearer distinctions between fence types has been, and still is, present

tion MathML standard which recognizes these three pairs as the only grouping operators standardly used in present-day mathematics.[5] There exist other pairs of standard opening and closing lexemes in various domains of mathematics, which act as circumfix (or parts of transfix) operators with a specific semantics. We will discuss them in detail in Section 4.1.6.3.

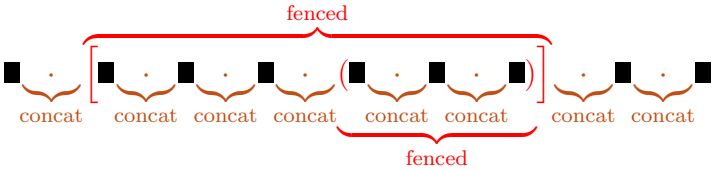
By design, fences are unambiguous syntactic cues that allow a reader to easily parse a mathematical expression, disambiguating contextual concerns such as:

1. precedence: $2(x+1)$ and $2x+1$
2. scope: $\sin(x/y)$ and $(\sin x)/y$
3. dominance: $\left(\sum i\right) + \left(\sum j\right)$ and $\sum\left(i + \sum j\right)$

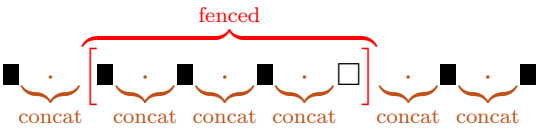
Syntactically, recognizing a fenced expression requires only a context-free notion of well-balanced symbol pairs. The fenced subexpression can be recursively treated as a stand-alone grammatical expression, and we would be wise to also expect the respective full range of phenomena. Hence, building on our first concatenation rule, we can state:

Concatenation Rule 2 Concatenation operators are to be discarded immediately after an opening lexeme and immediately before a closing lexeme, in the cases of fenced expressions, as well as circumfix and fenced transfix operators (Section 4.1.6.3).

Our running example can now be seen as:



We have taken for granted that mathematical expressions can be analyzed and parsed compositionally, as compositionality is now a widely accepted principle of natural language, but also in symbolic mathematics. To this extent, we remark that we will use a convenience notation for grammatical subexpressions, namely the symbol □. To illustrate, our example can be rewritten both as



in hand-written mathematics, where it becomes exceedingly hard to visually detect bracketing scopes in large expressions.

[5] As a side note, an alternative to using fences for grouping is to rely on operator dominance. [SW06] exemplifies that $3\overline{a+b}$ is an old notation for $3(a+b)$, used for the same purpose.

and further to

4.1.5 Mathematical Constructs

We proceed to examine the grammatical constructs in mathematical symbolism that do not expect additional arguments, but have an atomic syntactic category. In other words, the terminal subtrees in the operator tree of an expression. We consider both simple objects formed by single lexemes, as well as complex ones, induced by object-constructing operators.

We distinguish between two relevant properties for our exploration – an object's type and its structure.

4.1.5.1 Types

As we specifically remain interested in the logical structure of mathematical expressions, we will not consider the hierarchy of mathematical objects in full detail. Consider that each mathematical theory that defines its own spectrum of abstract objects, simultaneously establishes new types for its lexemes. Having a wealth of mathematical domains and a vast number of mathematical theories, we see that present day mathematics has an extremely rich taxonomy of object classes (or types). Examples would be sets, groups, vector fields, etc.

Being induced by theories, it is clear that most of this information is domain-specific and is not available in an underspecified setup. In this respect we would need to either completely discard it, or find very course-grained distinctions that we can observe via syntactic cues.

When approached in this manner, this is not a novel concern. We can directly adopt the two classical first-order logical types – those of terms (e.g. 1, $\mathbb{N}$) and formulas (e.g. $1 < 2$, $x \in \mathbb{N}$).

4.1.5.2 Structures

In parallel to a type, a construct also has a specific structure, that is entirely type-independent. The examples we are familiar with are:

1. object: $10m/s^2$ and $\mathbb{N}$
2. application: $1 < 2$
3. sequence: $1 < a, a < 2$ and a, b, c

Types and structures could be syntactically inferred, when relying on well-established conventions for using the various mathematical operators. This is the case since, as we will see, there is a clear separation between operators that act on terms, and such that act

on formulas. Furthermore, establishing the type and structure of a construct allows for disambiguating additional components of an expression subtree.

4.1.6 Mathematical Operators

Another universally adopted group of lexemes is mathematical operators. Most generically, operators are special mathematical symbols that are not syntactic atoms, but instead require to be provided with a number of arguments in an operator context. We distinguish between an "operator context", which is the standard application of an operator, and contrast it with an "element context", in which the operator symbol is used as a syntactic atom. An example of the latter is the group constructor: $(G, *)$.

When applied, operators are the verbs of mathematical expressions, and carry a range of special syntactic properties that allows them to be used with brevity and clarity by the working mathematician.

We present a categorization of the independent features we have observed, based on the orthogonal properties of type, order and fixity. Further properties we discuss are arity, dominance and precedence.

4.1.6.1 Types

Having established the base types of mathematical constructs in Section 4.1.5.1, we observe four classes of operator types, which we illustrate in Figure 4.1.6.1:

1. term operators – take one or more terms as arguments and construct a term.

   When clear from the context we will drop the "term" classifier.

2. modifiers – take a term and a formula as an argument and construct a term

3. relations – take one or more terms as arguments and construct a formula

4. meta-relations – take on ore more formulas as arguments and construct a formula

We admit that this distinction is as course-grained as one could allow, if we are to consider typing information. It hence fits exactly our knowledge-poor view on mathematical symbolism.

Another important remark is that we avoid the phenomenon of "framing" of mathematical objects by using this treatment. Framing is a well-known practice in mathematics where a mathematician decides on the type of an object based on their current contextual point of view. To exemplify, 1 can be seen both as a "number" and as a "set", while composition ($\circ$) could be seen as a "second-order term operator" or a "set". However, it is never the case that an object can be framed as a term and as a formula at the same time, allowing to avoid the induced pitfalls.[6]

[6]The author is aware that Gödel's Incompleteness theorems introduce framing between terms and formulas. However, there are a select few works of mathematics that use the technique, while the proposed typing system suffices for practical purposes in common math discourse.

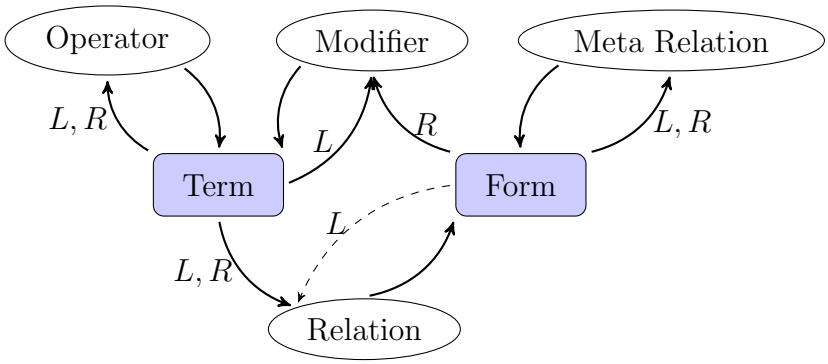


Figure 4.1: A Classic Type System for Mathematical Symbolism

4.1.6.2 Order

Order is a well-known property in type theories, used to build a hierarchy of operators based on their expected arguments. Atomic objects are zeroth order (e.g. $1, 2, \ldots$), functions are first order (e.g. $f(x), \sin(x)$), composition is second order (e.g. $f \circ g(x)$), etc. While there exists an infinite ladder of orders, standard mathematics typically does not go beyond second order operators. There also exist symbols that are flexibly overloaded to fit contexts of arbitrary order, namely equality ($=$) and some modifier symbols such as colon ($:$).

Due to framing, order can sometimes cause structural ambiguity, for example when viewing composition as a second-order operator or a zeroth-order set. For a human reader, this is usually clear from the internal expression context.

4.1.6.3 Fixity

With fixity we refer to the fixed argument patterns for applying an operator symbol. We have recognized prefix, infix, postfix, circumfix and transfix notations and will discuss them individually, as they play an important role in determining the logical form of an expression.

Note also that many operator symbols get reused in different mathematical domains and communities, which potentially induces new syntactic and/or semantic behaviours. In this regard, there are symbols that we have observed to have multiple possible fixities, which additionally contributes to structural ambiguity. For example, minus can be both prefix (e.g. unary minus: -1), infix (e.g. arithmetic minus: $2 - 1$) and postfix (e.g. grades/degrees: $A-$).

When known, fixity helps disambiguating an expression's syntax, in combination with precedence and all other properties of the expression constructs. However, when fixity is unknown, it contributes to the ambiguity of the syntax tree, as is the case for operators with multiple fixities or for fully underspecified lexemes.

Prefix operators precede their arguments, typically expecting a single argument to their immediate right. Examples are most standard notations for relations and functions, such as those in arithmetic (e.g. -1, $\sum i$), trigonometry (e.g. $\sin x$, $\cos x$), etc.

Infix operators expect two arguments, one on each of their sides. Examples are function applications (e.g. a/b, $a < b$), modifiers (e.g. $f : \mathbb{N} \to \mathbb{N}$), delimiters (e.g. a, b), etc.

Postfix operators follow their arguments, typically expecting a single argument to their immediate left. Examples are currencies (e.g. 100$) and factorials (e.g. 4!).

Circumfix operators dominate their arguments, either via surround them with a pair of opening and closing fence symbols, or by syntactically dominating them. Examples of such operations are:

1. mathematical functions: absolute value $|x|$, set cardinality $|S|$, lie bracket $[\cdot, \cdot]$, bra $\langle\phi|$ and ket $|\phi\rangle$ in quantum physics, ceilings $\lceil\cdot\rceil$ and floors $\lfloor\cdot\rfloor$, square root $\sqrt{x}$, logical negation $\overline{A \wedge B}$

2. object constructors: linear hull $\langle S\rangle$, group $(G, *)$, tuple $\langle x, y : x < y\rangle$, set $\{1, 2\}$ etc.

We observe that besides overloading the classical parenthesis, square bracket and curly brace fences, notations also use other well-balanced pairs. Furthermore, quantum physics presents one famous example of retailoring the use of angular brackets and bars to illustrate domain-specific semantics (such as orientation) of the operator.

We remark that while most classical fence symbol pairs can be handled in an underspecified setting, as they are commonly adopted in most domains, many notations exist that refactor the notion of well-balancedness and should be accommodated as exceptions.

Transfix (or mixfix) operators are all interplay patterns between and operator and its arguments that are too complex, or too irregular, to fit in the previous categories. They generally come in two flavours:[7]

1. fenced transfix operators: set constructor notations $\{\blacksquare \underbrace{\mid}_{\text{modifier}} \square\}$, $\{\blacksquare \underbrace{:}_{\text{modifier}} \square\}$ various interval notations e.g. $]\blacksquare, \blacksquare[$ or $[\blacksquare, \blacksquare)$

2. unfenced transfix operators: intervals $\int \square \, d\blacksquare$, quantifiers $\forall \blacksquare . \square$, typing judgements $\blacksquare \vdash_{\blacksquare} \blacksquare : \blacksquare$

[7] As transfix operators have complex argument patterns, we do not provide concrete examples from math discourse, but instead the generic patterns themselves, in order to clearly distinguish the operator components.

As with the circumfix notations, these operators also come in two semantic flavours – as applications of mathematical functions, or as constructors of objects. One could also think of circumfix operators as a specific kind of transfix operators. To exemplify, there is a thin line between the two when a transfix operator consists of well-balanced fences and delimiters, as is the case with intervals. There is not a clear distinction why intervals should be considered transfix, while tuples should be circumfix, and we do not insist on our suggested boundaries in these cases.

4.1.6.4 Arity

An operator's arity is the number of its expected arguments. Arities can be fixed (n-ary) or variable (flexary). In simple cases, they do not cause structural ambiguity, as fixity disambiguates the argument sequences of an operator. Infix operators mandate two arguments on each of their sides, prefix and postfix operators typically take a fenced argument sequence, or have other syntactic cues, such as using a different letter case for the operator and for the arguments (e.g. $Rxyz$). Circumfix and transfix operators typically dominate their arguments in an unambiguous manner.

4.1.6.5 Dominance

In our view on mathematical symbolism, a dominating operator always creates its own terminal subtree, in parallel of the main expression. For example, $a + 2\sqrt{x}$ would translate into $\blacksquare \underbrace{+}_{\text{infix op}} \square$.

$\sqrt{\cdot}$ is an operator which dominates its argument in a syntactically unambiguous manner, where both its argument and the resulting application can be handled compositionally. In this respect, operators with clear dominance are great disambiguation cues, as is also the case with two-dimensional fractions $\frac{\cdot}{\cdot}$.

Dominance is not always clear from the syntax, however, as is the case with summations. In $\sum i + \sum j$, it is not clear whether the first summation dominates the second or if they are on the same expression level. In order to disambiguate this particular example, however, one would need to understand at least which variables are bound by the summation operators, and most probably be familiar with the context in which the expression was used. We do not consider this to be a problem one can solve in an underspecified setting, and would consider it a valid example of structural ambiguity that should be preserved.

4.1.6.6 Precedence

Each mathematical domain, and often mathematical theories, defines syntactic conventions that ease the authoring of mathematical expressions and alleviates the requirement of fencing every applicative subexpression. These conventions typically come in the flavour of designating a strength of symbol binding, or operator precedence. In arithmetic, we know that plus ($+$) has weaker precedence than times ($\times$) or division ($/$,$\div$), which makes writing $2 \times x + y$ unambiguous. In logic, $\neg A \wedge B$ is also unambiguous as negation is known to have

higher precedence than conjunction. Such conventions exist in most domains that have rich collections of operators, but are impossible to know in a knowledge-poor environment.

Also, we should remark that knowing precedence is not always sufficient, as the order of application of operators of equal precedence (but different domain!) is sometimes only clear from an expression's context. A typical example is $\sin x/y$, which could denote $(\sin x)/y$ or $\sin(x/y)$, when disambiguated by fences. Note, however, that this is a problem that occurs when we juxtapose a trigonometric and an arithmetic operator. In the cases of two operators with equal precedence from the same domain, the ambiguity is no longer present, as western mathematics uses a convention of left-to-right evaluation of expressions. Namely, $1 + 2 - 3$ always denotes $(1 + 2) - 3$ and never $1 + (2 - 3)$.

An operator's domain, and more deeply – its semantics, plays an important role in restraining and selecting the appropriate precedence conventions. Units are an interesting case to examine: $10m/s^2$ is one of the rare examples where an invisible concatenation binds weaker than an infix operator, namely the unit composition "per" (/). If m and s were arithmetic parameters instead, we would typically evaluate $10m$ first, following the left-to-right convention. However, these examples are typically misleading, as they combine too many equivalence properties – equal precedence, same domain and associativity of arithmetic operators. Namely, while the most convenient reading of $10x/y$ is "ten times, x divided by y", it is equivalent to the reading obtained by following left-to-right evaluation: "ten times x, divided by y", which is a reliable cross-domain rule.

Two operators with the same precedence and domain could also induce ambiguity. Take as an example $-2!$. Only given the typing information that factorial is defined on positive integers, we can disambiguate the expression to $-(2!)$. Without this information, both readings are possible.

It is also important to remark that the precedence of visible operators and their invisible counterparts are not intuitively the same. As an example, $2 \times 4 + \frac{1}{2}$ binds differently from $2 \times 4\frac{1}{2}$. One is tempted to conclude that invisible operators bind stronger than their visible counterparts, which is true when the operators are of the same mathematical theory and no further conventions exist. Note, however, that this is not so reliable an observation in expressions with functions from different theories, as was our example with $10m/s^2$.

We presented a brief discussion of operator domains (or theories) together with precedence, for two reasons. First and foremost, precedences are defined within certain domains, which makes their use legit only after it is established that an expression belongs to a particular mathematical theory. Second, combining operators from different domains in a single expression induces interesting structural ambiguities, which is generally resolved via considering the deep types of the subexpressions, as well as the intra- and inter-expression context.

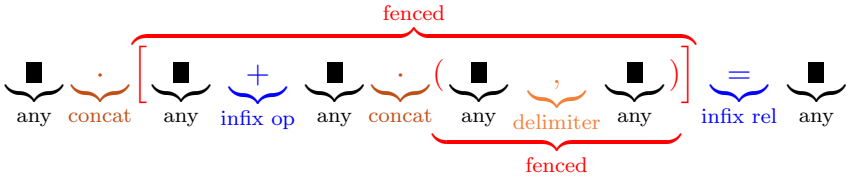
This concludes our study of operator properties.

4.1.6.7 Operator Wrap Up

To conclude, let us update our understanding of concatenation with a third rule, motivated by our decision to not attribute semantics to purely syntactic attachments:

Concatenation Rule 3 Concatenation operators are to be discarded immediately after a prefix operator, before a postfix operator and on both sides of an infix operator.

Taking these observations into account, we update our running example to:



(4.5)

4.1.7 Delimiters

Let us briefly look into two flavours of mathematical operators. Delimiters are symbolic operators that act as separators of subexpressions. They typically construct structural sequences(e.g. a, b, c or $a; b$) or act as terminators with various semantics – a dot(.) can be used to mark the end of an expression or to denote an abbreviation of an object's name. There is a wide variety of semantics corresponding to the structural sequences mentioned above – argument sequences, enumerations, orderings, progressions, etc.

An important note regarding the syntax is that delimiters are type-preserving, only changing the structure of the subexpression, as discussed in Section 4.1.5. To contrast, when a delimiter is used to make a sequence of a term and a formula (e.g. $1+1, a < b$), we consider this to be an "expression sequence" which is one meta level above the mathematical expression and the delimiters should have been written in the natural language modality. As this is a rule that authors do not consciously care about, writing presentational signs, we suggest that it could nevertheless be accommodated with a type "expression", parent of both "term" and "formula", which could be explicitly formed only for this use case.

4.1.8 Modifiers

Next, we consider modifiers. Grammatically, they act as "such that" or "which is" phrases and do not alter the overall expression tree, but instead attach additional information to a node of interest. As two examples, consider the set of pairs constructor $\langle a, b : a + b > 0 \rangle$ and the joint probability $P(a > 0, b > 0)$.

The first example is rightfully a transfix operator, but in the absence of domain knowledge its compositional interpretation is that of a circumfix operator with a modified argument. Here colon (:) modifies the sequence a, b with an arithmetic restriction. We can already notice the attachment ambiguity of colon, namely whether it modifies b or the entire sequence.

In the second case we use an infix relation as a modifier, similarly to a dependent clause. While this would typically give rise to a lot of type-related ambiguity, we remark that this particular notation is mostly (if not solely) used in sequential structures, which is a useful restriction to exploit. The mathematical semantics of the modifier clauses are typically range restrictions, elaborations, or typing operations.

4.1.9 Overall Assessment

The take-home messages from our structural analysis of mathematical phenomena span a broad spectrum. The most important conclusion is that if we want to avoid a combinatorial explosion of ambiguous parses, we can make due only with a holistic treatment for all standard properties of mathematical symbolism. The properties we consider particularly important for disambiguation cues are:

1. compositionality of mathematical symbolism
2. subexpression types (term, formula)
3. subexpression structure (object, application, sequence)
4. left-to-right evaluation
5. fixity, type and order of mathematical operators
6. binding and precedence of invisible operators
7. anticipate and accommodate all sources of ambiguity that can not be disambiguated by syntactic cues (interplay implicit via precedence, cross-domain and inter-expression context)

If we look at the final form of our running example in Eq. 4.5, we see that it is completely unambiguous if we consider the typing information (concatenation operators should only be term operators) and had a cross-domain rule of concatenation having higher precedence than plus. The latter is a specific case of the general rule that concatenation should bind stronger than any infix operator, which is what we have commonly observed, as long as the operators are from the same domain.

4.2 Symbols in Mathematical Expressions

We have established a hierarchical understanding of the structural phenomena out there, and are now ready to take an in-depth look into the special and alphabetic symbols used in the symbolic semiotic resource.

As we have decided to examine LaTeX corpora of scientific documents, we are first and foremost interested in the mathematical constructs that LaTeX provides. The best exhaustive survey of LaTeX symbols and expressions from a generic perspective which we have found is Wikipedia's help page on displaying formulas [For07]. It additionally provides a detailed bibliography of related surveys. A partial overview of mathematical symbols is also provided at [Wik]. As there exists a multitude of similar resources, our goal is not to systematize all of them together, but to try and generalize the structural behaviours of the more prolific symbols, as well as a classification for the less common ones.

Below we provide a small appendix of notable operators and their varying syntactic uses. We take a different points of view on the symbolic properties, as suitable for each case.

4.2.1 Vertical Bar: |

Prefix	Infix	Postfix	Circumfix	Transfix
n/a	$8\|4$ divisible by $X \mid w$ manifold constructor	$+\|_{U\times U}$ scripted restriction $x\|_a^b$ integral bound	$\|x-y\|$ absolute value $\|12\|$ determinant (lin. alg.) $\langle\phi\|$ and $\|\psi\rangle$ bra and ket (quant. ph.)	$\{x \mid x \in \mathbb{N}\}$ set constructor $\langle\phi\|\psi\rangle$ bra-ket (quant. ph.)

In our studies, vertical bar has proven to be the most overloaded operator with respect to fixity – the only fixity we have not observed is prefix. It is remarkable that a variety of seemingly unrelated mathematical domains have adopted their own notations

4.2.2 Equality: =

Equality has established itself as a universal infix operator, together with many of the related relation and meta-relation symbols. Interestingly when we discuss equality we see it shares a lot of its special properties with related symbols such as $\equiv$, $\approx$, $\propto$,$\simeq$, $\sim$, $\neq$, as well as with some of the arrows: $\Leftrightarrow$, $\leftrightarrow$ and the list continues. As mentioned above, we will not enumerate all symbols already classified in the exhaustive review articles, but focus on some irregularities in special cases.

What is special about equality is that while it has a constant fixity, it is extremely flexible in its order and type. Having adopted a classical typing system, it is hence "dual" in being both a relation (e.g. $1+1=2$) and a meta-relation (e.g. $\text{True} \wedge \text{False} = \text{False}$) of arbitrary order (e.g. second order: $f \circ (g \circ h) = (f \circ g) \circ h$).

Type duality is actually common in most equality-related and arrow operators (e.g. $\uparrow$,$\downarrow$,$\Uparrow$,$\Downarrow$,$\leftrightarrow$), which are also possibly dual in their order (although we have not encountered examples). We attribute this to the genericity of the denoted semantics of these symbols – the concepts of equality and map (and their fine-grained variations). We discuss arrows in further detail in Section 4.2.9.

4.2.3 Common Relation Symbols

To contrast with equality and arrows, there is what the author believes to be "common" relation symbols, that always behave regularly. Some notable examples are relations denoting membership (e.g. $\in$), order (e.g. $\geq$) or divisibility (e.g. $16|4$).

We can observe that common relations are uniformly infix and rarely if ever have dual types or orders.

An interesting phenomenon is that relation symbols can be used both as relations and as modifiers, to the author's convenience. An example would be $\{x \in X, y \in Y\}$ or $P(a < 0, b = 3)$. In our experience, all relation symbols can adapt themselves to modifier operators.

We also observe a flexible structure of the accepted arguments, in examples such as $a, b, c \in S$ or $a + b < 3$.

Another authoring technique is chaining of (possibly different) relations, as in $1 < a < b = 5$. This calls for an extension to our typed view on relations, as to allow a chaining with a preceding relation-formed formula, with the usual left-to-right evaluation.

4.2.4 Ellipsis: ... and ⋯

Ellipsis are typically syntactic atoms that act as an argument in a sequence or application structure. The ellipsis symbol is always three dots positioned either at the bottom, at the center or at a diagonal of the baseline. Consequently, it is dual in type as one could elide both term (e.g. $1, 2, 4, \ldots, 1024$) and formula (e.g. $A_1 \wedge A_2 \wedge \cdots \wedge A_n$) structures.

Ellipsis could also vary in order – above we showed zeroth-order examples, while a first-order one would be $f_1 \circ f_2 \circ \cdots \circ f_n$.

Fixity is no exception to the flexibility of ellipses, as they can be prefix (e.g. $\ldots, -2, -1, 0$), infix (see above) and postfix (e.g. $1, 2, 3, \ldots$).

4.2.5 Central Dot: ·

When used individually, a central dot can be used to denote an argument placeholder (e.g. $\sqrt{\cdot}$), as well as various types of products (dot product, arithmetic multiplication, matrix multiplication)

When used as an argument, it has infix fixity, first order and (to the best of our knowledge) only term type.

4.2.6 Lower Dot: .

The lower dot has a variety of syntactic uses. It is typically used as a postfix delimiter, or as a part of a transfix operator. In some notations it is also used as an infix operator, to denote function application, e.g. an absolute value $abs.x$[8]. A more radical example is Andrews' dot notation, in which a dot is used to fence the expression starting at the delimiter up to its very end. Nevertheless, it is possibly well-captured by the transfix operation view, as the dot notation typically accompanies variable binders, such as λ in Simple Type Theory and is indeed a domain-specific convention.

It is tricky to classify the syntax and semantics of cases where a lower dot is used to indicate abbreviations (e.g. *rad.* for radical). We could either consider it as part of the lexeme, or as a postfix operator that maps the abbreviation to its meaning. The author is uncertain what the best treatment would be.

[8]Seen at [Lib10]: `http://wiki.math-bridge.org/display/ntns/abs`

4.2.7 Quantifiers: $\forall, \exists$

Logical quantifiers always bind a variable which they quantify over. This induces transfix argument patterns that different notations have in different flavours. Here is a list of some of the patterns we have seen:

1. $\forall \blacksquare : \square$
2. $\forall \blacksquare . \square$
3. $\square, \forall \blacksquare$
4. Stacking: $\forall \blacksquare \exists \blacksquare . \square$[9]

The constructed quantified expression is always a zeroth-order formula in classical mathematical settings, but could be of arbitrary order if we were to analyze type theoretic expressions, such as lambda binders.

The bound object could be an atomic variable but also a list of variables (in our setup – a sequence structure).

4.2.8 Big Operators: $\sum, \prod, \bigcap, \bigcup, \bigwedge, \bigvee$

Big operators (to which we can also consider the integral) are characterized by a prefix big symbol operator that denotes an operation to be performed over a range of values for a bound variable. Usually, the ranges are provided in the scripts of the big symbol, or left implicit to the context (which we call "argument suppression"). The fixity is typically prefix, but transfix variants also exist.

They vary in type (term operators or meta-relations) and are usually first-order on the baseline, similarly to function application. One of the arguments is a variable bound to a range, and typically resides in the scripts. Still, some notations allow range argument to appear on the baseline (e.g. $\sum \blacksquare : \square$), or to at least denote the bound variable (integral notation).

4.2.9 Arrows

In Section 4.2.2 we already noted on the type duality and possible multiple order of the arrow operators. Some observations we have made, mainly through the resources used to compile Libbrecht's notation census [Lib10] follow:

1. $\downarrow$ and $\uparrow$ stand for "Nor" and "Nand" in in proof theory, and are thus infix meta-relations.
2. $\downarrow$ could denote "smallest element" as a prefix term operator

[9] Sometimes there is an additional lower dot separator between the stacked quantifiers, but this varies across notations.

3. $\uparrow$ could denote "bigger-of" as an infix operator (e.g. $abs.x = x \uparrow -x$.)
4. $\Rightarrow$ typically denotes logical implication, an infix meta-relation
5. $\rightarrow$ is typically a type constructor, or map, an infix term operator, but is also used as a meta-relation.

4.2.9.1 Plus and Minus: $+$, $-$

We have already remarked on the ambiguous fixity (prefix, infix or postfix) of plus and minus in the beginning of Section 4.1.6.3. Apart from that, they are consistently term operators in an application context and, just as any other operator symbol, are permitted to be used as syntactic atoms in element context.

4.2.9.2 Colon: :

Colon has two common uses – an infix operator denoting division (e.g. $\frac{1}{2} : \frac{3}{4}$[10]) and an infix modifier, typically introducing types (e.g. $f : \mathbb{N} \rightarrow \mathbb{N}$) and "such that" restrictions (e.g. $\{x \in \mathbb{R} : x > 0\}$).

4.2.10 Notation variants of the same content objects

The purpose of this section is to briefly illustrate the many-to-many relationship between syntactic signs and implied semantics. We have already shown that a single sign can represent a variety of denotations and will now illustrate the other direction.

[SW06] lists a variety of notations for partial differentiation:

$$f_x,\ f'_x,\ d_x f,\ \partial_x f,\ \nabla_x f,\ \frac{\partial f}{\partial x},\ D_x f$$

We have also seen examples with absolute value ($|x|$ and $abs.x$), division ($/$, $:$ and $\div$), multiplication ($\cdot$, $\times$) and the list continues.

This reaffirms our intuition that an understanding of a mathematical expression needs to be contextualized to its mathematical subdomain, together with its concrete community of practice, in order to avoid a blow-up in structural ambiguity. Additionally, the definition contexts within each particular document bring relevant context that may alter notations and introduce new lexemes or denotations. If we were to adopt an alternative where we substitute ambiguity by underspecification, we would quickly find ourselves into a fully underspecified setting, with an induced logical form that has hardly any structure, and hence no real contribution.

With this we conclude our exploration of structural phenomena and will continue to face some of the uncovered challenges, in the next chapter.

[10] Seen at `http://wiki.math-bridge.org/display/ntns/divide`

Chapter 5

A Grammar for Mathematical Symbolism

We proceed with pursuing a solution to understanding the logical structure of symbolic mathematics, from a large-scale perspective. Our approach is developed with and designed for real-world corpora of scientific publications.

In the terminology of Math Recognition [ZB11], the signs of mathematical symbolism are visually ordered into "layout trees" and the "extraction" (or recovery) of structure is a consequent step leading to the creation of "operator trees".

The author has chosen the domain of analysis to be that of Western one-dimensional inline expressions in scientific documents, as vast digital resources exist in that area. To this extent, currently scripts are treated in a preprocessing step, being compositionally detached and later reattached to their base lexeme, remaining fully underspecified. We consider that they would nicely fit in an expanded grammar that also considers domain knowledge, which is an interesting direction for future work.

In existing digital corpora of LaTeX documents, such as arXMLiv, the reliable level of explicit structure is indeed that of layout trees. This observation is deemed uncontroversial, as the process of writing corresponding LaTeX markup explicitly encodes the presentational layout of symbolic signs. This is why the author considers layout trees as the viable starting stage of analysis for the recovery of logical forms.

5.1 To Pose a Problem

The ultimate goal of a computational process of understanding mathematical expressions would be to to extract an exact logical representation of the intended semantics, achieving both a correct structure, as well as mapping each sign to completely unambiguous semantic counterparts.

We have provided a range of examples in Chapter 4, which demonstrated the various challenges that need to be overcome, in order to truly recover the semantics of an inline mathematical formula. We have also demonstrated in Chapter 3 that mathematical notation is highly adaptive and flexibly allows overloading and redefinition of its lexical

entries.

To this extent this thesis will not attempt to solve the full task of understanding mathematical symbolism – instead it only tries to pose the problem.

As it turns out, and we have for the first time clearly seen this in Chapter 4, there are really two stages to our task – establishing the logical structure and establishing the exact semantics of lexemes. And while the latter task is a challenging, domain-specific and context-dependent analysis process, recovering the tree structure is largely syntactic. It would be safe to say that if one provides all complex notations as given by their domains (e.g. transfix argument patterns), the operator tree of mathematical expressions could be reliably built using solely syntactic information, leaving open only ambiguity issues pertaining to operator precedence and scope.

It is hence the experimental aim of this thesis to seek the limits of syntactic parsing methods in a knowledge-poor setting and to test the intuitions developed thus far. For this goal we can relax our desired outcome, and attempt to extract an underspecified superset of the intended operator tree of each input expression.

Our vision of a full solution, would encompass an adaptive approach (e.g. learning from discourse definitions), based on dynamic tools (e.g. a grammar formalism capable of adapting to new lexical knowledge in runtime) – ideas that have just started to be explored by Ganesalingam [Gan09] and the FMathL project [KN11].

5.1.1 Presentation Input Formats

In order to have a reliable process, we require a linguistically neutral starting point, i.e. a document which avoids making any assumptions that were not explicitly provided by the author. This point is discussed in detail in [GJA+09].

However, this is a requisite that can be fulfilled by virtually any input representation. We fix our input format to the *.noparse.xml* format preferred by [GJA+09], but remark that it is potentially trivial to normalize various input representations towards it, such as Presentation MathML. There already exist transformation workflows for TEX/LATEX, but one could enable workflows from virtually any visual image (via Math OCR), as well as from modern formats such as Presentational MathML, as there are developed tools for their respective conversion into LATEX. In the field of Math Recognition, the construction of "layout trees" is considered a largely solved problem and such trees are the exact representation of mathematical expressions in *.noparse.xml*. This gives us confidence that our starting point has the potential to scale to the full legacy of retro-born-digital scientific documents and beyond.

Generically, the requirements of an input representation are to strictly follow the original source, in the sense that all signs occurring on the base line would be captured as a "linear sequence of tokens", while any optional, but unambiguous, structural or semantic information encoded by the author will be additionally interleaved.

The *.noparse.xml* format meets these conditions and could suggest that further interactivity on top of the initial input could be provided to the user by an intuitive interface, e.g.

so that the author could dynamically disambiguate his presentational input. We discuss this idea in more detail in Section 10.1.2.

5.1.2 A Lexer for Mathematical Symbolism

As our approach trusts a layout tree given on input to be adequate, this subsumes that one must also trust an underlying lexer which was used to create that tree. For the experimental part of this thesis, we use the representation created by the LaTeXML software and will hence rely on its lexical rules for mathematical symbols – e.g. to distinguish between term operators and relations, between big operators and quantifiers, etc. While we acknowledge the particular lexer isn't perfect, is already reliably covers all native LaTeX symbols, together with those in some of the most used style packages.

We would in principle propose to improve the lexer to cover all properties and subcategorizations that we discussed in detail in Chapter 4, and would consider this as direct future work to the current thesis.

5.1.3 Errors in Mathematical Expressions

It is only to be expected that large-scale corpora of uncontrolled scientific documents will have some degree of errors in their mathematical expressions. There are many reasons of why such errors could occur – problems of modality (e.g. sporadically leaving fences in different modalities, making the expressions unbalanced), typos (e.g. leaving a trailing punctuation mark, or even an operator at the end of an expression such as 1/2/3/), mistaken and omitted decorations or styles (e.g. $\tilde{x}$ instead of $\overline{x}$), etc.

We do not address these problems here, but still point them out when their instances occur in our experiments. The author can see a two-fold approach to resolving such errors, one being a knowledge-poor set of heuristics, such as the LaMaPUn Purification module [Gin11, LaMaPUn::Preprocessor::Purify module], another being a knowledge-rich approach to fault tolerant interpretation of formulas, such as [HW06a].

5.1.4 Compositional Treatment of Vertical Components

We remarked that scripts act as modifiers and hence do not alter the structure of the operator tree determined by the baseline expression in Section 4.1.2. We indeed avoid treating scripts as grammatical elements of our fragment of inline mathematical expressions, but will instead use a compositional treatment that operates by:

1. parse script contents as an independent expression
2. detach scripts from baseline symbols, achieving a linear form
3. after successful parse of the baseline expression, re-attach scripts
4. use fully underspecified semantics, e.g. via a generic script element, that makes no claims on the denotation of the scripted expression

Hence, this postpones the task of determining the script semantics either to a post-processing semantic analysis, or to a later, extended version of our proposed grammar.

The main goal of this treatment is to reduce the explosion of ambiguous parses (we re-attach but a single tree, which hides any ambiguity occurring within the scripted expression) and restrict the parsing problem to constructing an operator tree from a one-dimensional expression.

For the same reason, we need to treat the cases of subexpressions with explicit structure. Those are either two-dimensional fragments (e.g. matrices, fractions) or explicit markup for dominating operators (e.g. roots, binomial coefficients). For these cases, we substitute these objects with a generic lexeme that denotes a one-dimensional syntactic "anything" for the main parse of the baseline. We again process the subexpression in parallel and re-attach it back in its original position in post-processing. It is already clear that this also requires a post-processing pruning procedure that weeds out senseless parses.

5.2 Grammar Overview

As justified in Chapter 4, we constrained the scope of our analysis to linear sequences of tokens, i.e. a horizontal sequence of symbols positioned on the same baseline. In this setup, mathematical expressions could be parsed and analyzed similarly to natural language sentences. Naturally, that process would be governed by a different set of rules, i.e. a different grammar, and has its own lexicon and exhibited phenomena.

As mathematical symbolism has evolved following the primary principle of unambiguously conveying knowledge to the educated reader, we could rightfully expect that its signs provide a transparent syntax-semantics translation, close to that of programming languages. This transparency stems from well-defined conventions regarding the precise use and meaning of the various mathematical objects and their interactions, however specific to a given domain, or even mathematical theory. To capture this intuition, the author has chosen the Combinatory Categorial Grammar formalism (CCG) [Ste00], which provides a lexicalized approach to capturing the compositional syntax-semantics relationships of its lexemes. This will also prove useful when resolving different kinds of ambiguities, as one can easily incorporate the expected behaviour in a given domain, which as we argued above is intentionally unambiguous.

The formalism also provides a convenient way of modeling domain knowledge, used for knowledge-rich parsing, via grammar "feature" attributes. One also has the potential to be uncommitted to any domain and enable knowledge-poor parsing, which could coordinate the process via lexical categories, also utilizing intra-expression context.

Another of CCG's features with high-relevance for mathematical discourse are its multiple application modalities – we will see that its associativity modality is an excellent way to accommodate second-order interplay of mathematical operators.

5.2.1 Compact Disjunctive Logical Forms

The result of the extraction phase would ideally be a logical form, which represents an underspecified and potentially ambiguous mathematical expression. We are interested in building a fundament for further work, as well as to stay neutral to making uninformed assumptions, hence require the preservation and explicit representation of these phenomena.

To this end, we utilize a notion of "Compact Disjunctive Logical Forms" (CDLF), which captures the underspecified components via generic categories (e.g. "token"), while making ambiguity explicit via disjunctive forests of logical forms. We supplement this with a notion of "compactness", in order to reduce the size of the disjunctive tree, by packing together the identical subtrees. Both packing and unpacking algorithms have been developed and are presented in more detail in Chapter 9.

Such forms are widely studied in computational linguistics and are also called "shared-packed forests" [Har94].

5.3 Analytic Design

In Chapter 4 we established a cross-domain view on mathematical phenomena. That view directly translates into a layered approach to designing a Combinatory Categorial Grammar for math expressions, which we outline at both the analytic and experimental stages.

5.3.1 Lexical Categories

We directly mirror the classical view on mathematical objects presented in Section 4.1.5.1, by adding a "term" and "form" terminal categories. We also add a special "expr" category that is used solely as a sink for delimiters that take in arguments with mixed types. We accommodate structure via a hierarchical "STRUCT" feature family, described in the following section.

5.3.2 Feature Families

In order to use intra-expression context to disambiguate the possible parses of a given expression, we need to be able to take into account all individual properties that are exhibited by its lexemes. To this extent, we it is very convenient to use CCG's feature family capabilities, in order to define a range of features that can drive this process.

The current features built into the grammar are:

1. STRUCT: subexpression structure, follows Section 4.1.5.2
2. FIXITY: subexpression fixity, follows Section 4.1.6.3
3. GLUE: follows the glue considerations in Section 4.1.3

4. FENCE: well-balanced pair kinds for fences and circumfix operators, follows Sections 4.1.4 and 4.1.6.3.

5. CASE: used to capture syntactic cues for applications such as aRa.

6. ontology: contains a small hierarchy of underspecified mathematical objects

The author believes that in the long run there would be benefit in adding a "domain" feature, in order to easily incorporate knowledge-rich rules and properties (such as cross-domain precedence). Let us reiterate that the grammar is currently trying to incorporate cross-domain rules that are valid in arbitrary mathematical expressions, written in present day scientific publications. In that respect, any deviations from the standard behaviour are probably best treated as exceptions in the specific domain where they occur.

There is also a definite need to refine and reorganize the hierarchy for the structure feature, as the current version does not properly capture the author's intuitions yet.

5.3.3 Lexical Families

As CCG is a lexicalized grammar formalism, we need to account for all structural phenomena on the lexical level. We incrementally build up a system that models the core fragments of mathematical symbolism and accommodates for its various structural irregularities.

5.3.3.1 Cross-domain Parse Anchors

In a knowledge-poor setting, with no domain information, there is very high risk of super-exponential explosion of the parse trees with a growing size of the expressions. This would both make the process intractable and produce an extremely ambiguous logical form, rendering the approach worthless in practice. To avoid this, we take a critical view on the modeling process, pinpointing four anchors that act as strong disambiguation cues for our grammar. These are:

1. well-balanced pairs of fences or circumfix operators

2. type distinction between operators, modifiers, relations, and metarelations

3. accurate parsing of invisible operators – highest precedence among infix operators, awareness of argument structure

4. structure distinction in subexpression – correct contextual treatment of sequences, objects and applications

The author believes that when put together, a basis of these four principles goes a long way in reducing the structural ambiguity of the parses to an applicable degree. We will pay brief attention to the specifics of each lexical family.

5.3.3.2 Fences and Circumfix Operators

We allow for parenthesis to be used both for term and formula typed arguments, as well as for empty arguments. In order to handle them in a lexicalized grammar, we realize them via families of opening fences, e.g. of the form:

```
entry: term<~2>[angular expression noglue X0]/*close[angular X1]/*term<2>[noglue X2]:
       X0:open(* <Arg>X2 <Close>X1);
entry: form<~2>[angular expression noglue X0]/*close[angular X1]/*form<2>[noglue X2]:
       X0:open(* <Arg>X2 <Close>X1);
# Empty angular brackets:
entry: term[empty angular noglue X0]/*close[angular X1]:
       X0:open(* <Arg>empty <Close>X1);
```

The closing symbols of the pairs are in a respective "close" type, with a FENCE feature that specifies the opening constructor (e.g. "angular" for the pair of angular brackets $\langle\cdot\rangle$).

This treatment achieves parsing each fenced subexpression as separate from the main baseline tree, reducing syntactic ambiguity. However, we would in principle need to extend the family with further rules in order to accommodate and higher-order fenced subexpressions. The author is yet to encounter real world examples of such constructs, but I anticipate that they indeed exist in mathematics.

5.3.3.3 Concatenation

The grammar incorporates concatenation in its own lexical family that accommodates several properties:

1. it always has higher precedence than visible infix constructs
2. it should be able to attach itself as syntactic glue in cases such as aRa.
3. it could be the mechanism to support fenced modifiers, as in the case of $l(\geq q)$. This treatment could be replaced by allowing higher-order expressions within fences.

5.3.3.4 Operator Symbols

We start by creating families for the ground "term" and "formula" objects, with "atomic" fixity.

Next, the grammar provides categories for prefix, infix and postfix term operators, modifiers, relations and meta-relations. The special properties we incorporate are:

1. legalize use of operator symbols as atomic elements in sequence contexts ("element" structure with "atomic" fixity)
2. enable higher-order chaining of prefix operators, by using the associative modality ($\backslash\hat{}$) for their argument.
3. enable the chaining of infix relations following the left-to-right evaluation convention.

4. legalize the flexible use of infix relations as modifiers ("which is" statements)
5. provide special entries for second-order infix relations
6. distinguish between application and sequence contexts via the STRUCT feature
7. provide a family for term operators with ambiguous fixity (potentially needed for all operator types)
8. provide a category for second-order term operators, having in mind the specific case of the composition operator ($\circ$ or ;)
9. allow type duality of the right arguments of infix modifiers[1], e.g. both $a : \mathbb{N}$ and $a : a \in \mathbb{N}$ are grammatical.
10. provide special families for punctuation and sequence delimiters and accommodate expression-level list construction

5.3.3.5 Arrows

Currently, we incorporate a single lexical family for arrows, "ArrowDual", which treats the arrow symbols as infix term operators or infix meta-relations. Notably, we have not added support for the vertical arrows (such as $\downarrow$,$\Uparrow$), which besides being dual in type are also ambiguous in fixity (prefix, infix or postfix). Notably, this is an easy enhancement to our grammar, combining the treatment in the "ArrowDual" and "AmbigfixOp" families.

5.3.3.6 Anything

The "Anything" family covers all symbols that do not have a standard role in mathematical symbolism, such as alphabet letters or full words. Note that we do not consider unknown operator symbols as "Anything" and we shouldn't, as this would induce a too severe degree of ambiguous parses. This translates to us making the assumption that all operator, fence and punctuation symbols are known to our lexer and grammar in advance – a requirement we are willing to realistically pose due to the particular markup needed to input operator symbols in LaTeX.

5.3.3.7 Transfix Operators

In the current effort, we have tried to avoid the addition of transfix patterns in our grammar, including quantifiers and big operators. This decision was made due to the observation that these notations are defined in specific subdomains of mathematics and their notations additionally differ based on the convention in the "community of practice". The only exception is the support for the Russian and French interval notations, which however are on the border between transfix and circumfix fixities.

[1] We have only encountered infix fixity of modifiers thus far.

5.3.4 Generic Lexemes

To use the cross-domain applicability of our grammar in full generality, we try to stay as uncommitted as possible to the specific lexemes encountered in mathematical discourse.

The author currently has not tried to work on adapting the LaTeXML lexer, while it presents a few assumptions that are too narrow (e.g. = is considered a relational operator, but we have seen it is dual in being a relation and meta-relation). Additionally, the analysis in Chapter 4 reveals various properties that are yet to be addressed by the lexer (e.g. there is no "modifier" category). Hence, we relax our ultimate requirement of fully generic lexemes and provide specific rules for some of the symbols that require more special treatment, such as "=", "|" and ";".

Besides the benefit of keeping the formalism in a generalized state, the practical enhancement from using generic lexemes lies in reducing the diversity of the set of input expressions, making the memoising of parse runs more powerful. This translates into a much lower runtime over large collections of documents. We should keep in mind a caveat related to memoising parses, namely that it requires an interface hosting a strongly coupled map between the lexer output and the generic grammar lexemes.

5.3.5 Semantics

As our type system and lexical families are completely generic by design, this induces a similarly generic, underspecified logical forms on output. The grammar currently models the semantics as applications of function symbols to arguments, which is in direct correspondence with the XML representations of content mathematics.

However, the CCG formalism allows for this approach to scale in the future, as it can incorporate bound variable information and a more fine-grained ontology of mathematical expressions. As the inferred semantics of subexpressions can be directly used as features within lexical rules, this provides a powerful mechanism for simultaneously disambiguating expression syntax and semantics. While this is just a claim at the time of writing, the author is excited to investigate this further, together with incorporating domain information in the grammar.

5.4 Experimental Design

During the phase of experimentally developing the grammar coverage over our training document collection (described in Section 6.1.2), we encountered a range of phenomena that contributed to improving our grammar in order to be capable of handling real-world math expressions in actual scientific publications.

These improvements include:

1. fully testing the interplay between visible operators with all fixities, together with infix concatenation

2. allowing a fenced sequence to act as an application, i.e. to be taken as arguments of applicable operators and relations which do not expect sequence arguments (e.g. $G = (S, *)$)

3. accommodating framing, via rooting higher-order symbols as term elements in sequence context (e.g. in $f \circ g = \{\text{some set}\}$, composition would construct a list element)

4. support for the full spectrum of equality behaviours – assignment modifier, relation and meta-relation[2]

5. support for second-order equality and chaining of prefix operators (e.g. in $\partial_x^2 = \eta^{\mu\nu}\partial_\mu\partial_\nu$)

6. providing support for incomplete bracketed modifiers (e.g. $M_{21}^b(\neq 0)$)

7. support for empty groups, such as () or ||.

8. support for relations on argument sequences, e.g. $a, b, c \in \mathbb{N}$

9. support for $<$ and $>$ used as fences, instead of $\langle$ and $\rangle$. It seems that authors prefer them as being visually more appealing in certain situations.

We consider this to be only a rough first approximation of what the accommodated behaviours in mathematical symbolism should be and definitely see large room for improvement both in theoretical clarity and in practical fine-tuning of the suggested grammar rules. A full copy of the current grammar in the short syntax of OpenCCG is available in Appendix A.

[2]support is also needed for all equality-related operators (e.g. $\equiv$ and $\approx$), which also exhibit all three behaviours. This is still partial in the current grammar.

Chapter 6

Methods

For the experimental work in this thesis, we have relied on several document collections, which we introduce in Section 6.1. We describe our setup for developing and evaluating our grammar in Section 6.2. As processing multimodal XML documents is a task with a variety of representational challenges, we conclude with describing our exact implementation approach in Section 6.3.

6.1 Document Collections

As the work on developing our grammar was fist in a theoretical (or analytic) direction and later tested against a practical evaluation over a sandbox of document, we distinguish between two collection purposes:

6.1.1 Analytic Resources

We have surveyed the following resources as a basis for our description of mathematical phenomena (Chapter 4):

1. Michael Stoll's collection of Jacobs University lecture notes in: Algebra, Number Theory, Complex Analysis, Algebraic Geometry and Integration and Manifolds. [Sto]
2. Paul Libbrecht's Notation census [Lib10]
3. Elena Smirnova's notation selection tool [SW06]
4. The training sandbox of mathematical documents described below (Section 6.1.2).

The emphasis in this selection was put on broad coverage over various domains, while also reviewing in depth monographs in selected topics (e.g. lecture notes and scientific articles). The author's hope was that the selected works flesh out a big portion of the phenomena in mathematical symbolism, while being diverse enough to draw a clear distinction between domain-specific and cross-domain properties.

6.1.2 Experimental Resources

For the practical evaluation of our approach we created two small sandboxes of handpicked documents from the arXMLiv corpus [SKG+10]. Each sandbox abided by the following criteria:

1. Collection of 10 scientific articles

2. Each document is from a different scientific field, namely:

Training set	**Testing set**
Differential Geometry	Functional Analysis
Quantum Physics	Algebraic Geometry
High Energy Physics – Theory	General Relativity and Quantum Cosmology
Commutative Algebra	Metric Geometry
Statistics Theory	High Energy Physics
General Relativity and Quantum Cosmology	Analysis of PDEs
Cosmology and Extragalactic Astrophysics	Computational Complexity
Exactly Solvable and Integrable Systems	Differential Geometry
Geometric Topology	Probability
Algebraic Geometry	Statistics

3. Each document has demonstrated grammar failures in parsing with the LaTeXML software, but no other errors were present.

Again, we were interested into a diverse collection of scientific texts that is both big enough to be representative, but also constrained enough to be manually verifiable by the author alone. The third condition was set, so that we explicitly pose the challenge of addressing LaTeXML's limitations and also to investigate what the exact problems consist of.

6.2 Development Setup

The author developed the current version of the grammar for mathematical expressions in several stages.

First, a large number of documents and their expressions were surveyed in an attempt to extract the structural principles in their formation, which we have discussed in detail in Chapter 4. A small testbed of expressions was developed to support our expectations of what expressions are grammatical and to what extent they would be structurally ambiguous. That set of grammatical input expressions can be seen at Appendix A.4.

Second, the grammar was repeatedly tested against the training set, via the API provided in the LaMaPUn library [Gin11]. In this stage the author had to solve a number

of practical issues, related to the peculiarities of LaTeXML's XML representation and the information provided by its lexer. During the incremental improvement of the grammar coverage, the author compiled a second component to our expression testbed, available in the latter part of Appendix A.4.

As the experiments gave us further insight into the nature of the phenomena encountered and the limitations of the approach, this lead to both the repeated improvement of the grammar and of a better understanding of the grammatical interplay in expressions, described in detail in Sections 5.3 and 5.4.

Finally, we tested the final version of the grammar against the test set, the results of which we report in Chapter 7.

6.3 Implementation

While we have developed the grammar with the OpenCCG toolbox [CCG09], it has been integrated as a module component of the LaMaPUn library [Gin11], which we assume as a prerequisite in the discussion below. In fact, all experimental and implementation work has been performed in the context of the LaMaPUn library, which has also been created and maintained by the author.

6.3.1 A Unified Analysis Pipeline

Putting all components together, the extraction phase is compiled into a coherent analysis module integrated in the Semantic Blackboard section of the LaMaPUn library [Gin11], under the name "FormulaTreeParser".

A pass over an arXMLiv document consists of:

1. Providing a scientific document in source TEX representation on input
2. Converting the source to "*.noparse.xml*" representation using LaTeXML's *–noparse* mode.
3. Purifying the textual and mathematics modalities, via the "LaMaPUn::Preprocessor::Purify" module.
4. Retrieving all mathematical expressions in the document via the "LaMaPUn::Preprocessor" module.
5. Extracting the logical forms of each expression via the newly built "LaMaPUn::SemBlackboard::FormulaTreeParser" module.

Once the logical form parses have been constructed, they are ready to be processed by the representation modules, finally producing a cross-referenced Presentation plus Content MathML representation, ready to be used by external applications, such as formula search.

6.3.2 Representation Toolbox

We discuss the details behind the representation part of this thesis in Chapter 9. We have developed the API and algorithms using a variety of parse forests obtained by our grammar, but have not attempted a real evaluation at the time of writing.

Nevertheless, the first version of a toolbox exists in a functional, well-documented state. It is the author's commitment to maintain and continue to develop all modules in the LaMaPUn library, eventually supplying them with a detailed test suite, realized via one of Perl's fine testing frameworks, such as "Test::More".

Chapter 7

Evaluation

To evaluate our work, we test against a training and a testing document collection, introduced in Section 6.1.

While examining the expressions in these collections, let us adopt two groups of distinctions:

1. baseline and fragment expressions – we will treat separately the parses of entire baseline expressions, from the parses of their fragments, such as scripts or arguments of graphically dominating operations (e.g. fractions and square roots).

2. unique and cumulative expressions – it is useful to examine the differences between a document-centered view, in which one aims to cover as many real world expressions as possible, and the grammar-centered view, in which one wants to model the biggest number of unique grammatical expressions, regardless of their frequency in written texts.

On top of the traditional evaluation measures, we deviate to examine expression length, measured in the count of their symbols, as it is important to establish whether the number of lexemes comprising each expression influences the number of grammatical parses.

We are interested in maximizing several measures of the grammar's utility:

7.1 Recall

First, we examine the coverage of the grammar over all expressions present in the test and train collections. While the document count was the same, ten in each collection, the train set has over double the amount of expressions than the training set (over 12,000 versus less than 6,000). Such differences could of course be expected, as the number of expressions per document varies greatly with the intention and field of a given article (e.g. theoretical publications that prove novel theorems have over a magnitude more formulas than descriptive publications that report on empirical observations).

Interestingly, the unique baseline expressions were a fifth of the overall baseline entries, while the unique fragments were only one percent of their respective cumulative count. A detailed exposition is available in Table 7.1, which reports on the number of expressions in the collections, together with the frequencies of grammatical failures encountered while parsing with our grammar. Note that these documents are not annotated and should not be considered a gold standard, as we also make clear in our error analysis (Section 7.5) by enumerating several truly ungrammatical expressions present in the sources that were never previously expected.

	Train Set	Test Set
Total Baseline Expressions	12245	5821
Total Fragments	13378	5789
Unique Baseline Expressions	2091	1156
Unique Fragments	145	120
Failed Baseline Expression Parses	31	37
Failed Fragment Parses	30	2

Table 7.1: Expression Counts in Train and Test Collections

Thus, in Table 7.2, the author reports the following recall rates:

	Train Set	Test Set
Total Baseline Expressions	99.74%	99.36%
Total Fragments	99.77%	99.96%
Unique Baseline Expressions	98.5%	96.79%
Unique Fragments	79.31%	98.33%

Table 7.2: Recall Rates in Train and Test Collections

As we can see, the grammar achieves a very wide coverage on both train and test sets, despite its confessed limitations (outlined in Section 7.5). To contrast, the LaTeXML processing of these documents reports over 900 failed parses of baseline expressions, compared to a tenfold cumulative improvement of 60 failures in our CCG grammar.

There are probably three factors at work for this result:

1. the grammar has successfully incorporated the core cross-domain properties of mathematical symbolism

2. the expression sample is too small – less than 20,000 baseline expression overall, which equates to about 0.1% of all arXMLiv expressions

3. the genre of scientific publications somewhat restricts authors in not regularly using complex custom notations, as they are typically bound to a certain number of pages which must not be exceeded.

Another pitfall to beware in the percentages above are false positives – there might be many expressions that do not receive an appropriate parse tree (e.g. transfix operators such as integrals and quantifiers), but instead get successfully approximated by a more generic treatment (e.g. as prefix operators). Not to be seen as a contradiction, while the 18,000 expressions we have tested against are but a tenth of a percent of corpus coverage, they are still too numerous for the author to examine by hand.

In the following section we discuss the precision of these parses, paying attention exactly to false positives – logical forms that indicate a successful parse, but do not represent the semantics intended by the author.

7.2 Precision

As the expressions in our collections were too numerous, the author proceeded to manually inspect ten random successful parses from each document in the train and test sets. The observed parse forests were largely satisfactory (i.e. they were a superset of the intended parse by the author), with the following exceptions that require further improvement:

1. The type duality of equality (either a relation and a meta-relation) is still not accommodated, as only a single parse is generated for examples such as "$A \wedge B = C \vee D$".

2. All transfix operators are, by design, parsed as different kinds of constructs, which would benefit from future work that incorporates domain-specific and community of practice information.

7.3 Certainty

A major concern at the beginning of this thesis was to avoid combinatorial explosions of output parses, which would render the parsing process intractable and make the parse forests too vast to be of practical use. As we are dealing with underspecified inputs, this problem is essentially imminent, if not explicitly addressed. The design choices influenced by these concerns were to abide by a left-to-right evaluation convention, as well as to thoroughly deposit concatenation operators around all highly ambiguous lexemes.

The author has carried out an evaluation of the degree of ambiguity of the produced parses and made them available in an appendix to this thesis. Figures B.1, B.2, B.3 and B.4 present the frequency of grammatical parses over the given input expressions for both train and test sets. We distinguish between two types of expressions and a cumulative versus a unique overview, in order to grasp a better understanding of our coverage.

The results seem to be a very promising indication for the reliability of the approach. If we look at the unique expressions (e.g. Figure 7.1), we clearly see that the ambiguity

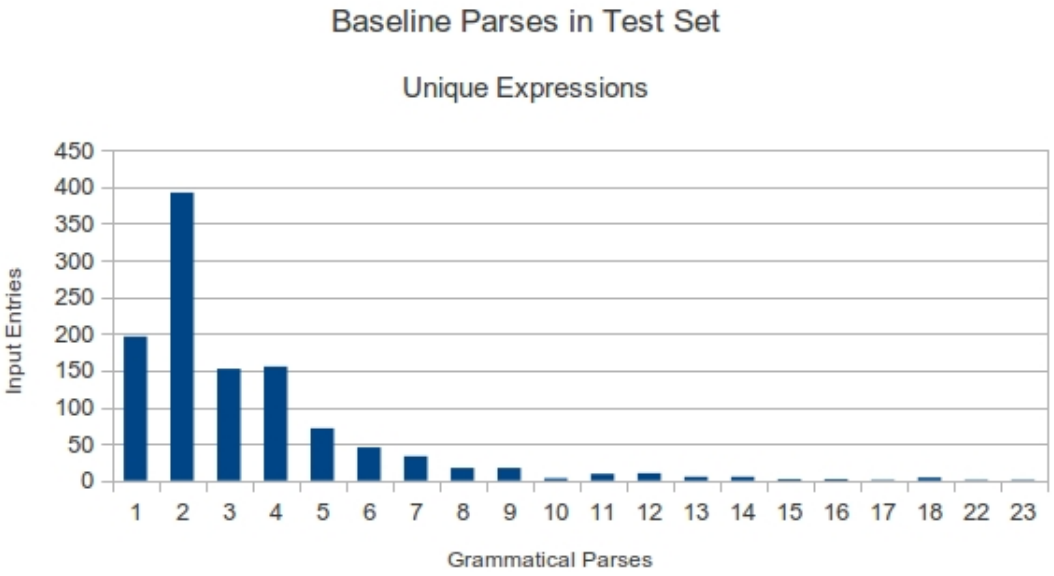


Figure 7.1: Number of Grammatical Parses over Unique Baseline Expressions in Test Set

levels are manageable and well-behaved. Most expressions have five or less grammatical parses, while the number of expressions with 10 or more parses is statistically negligible.

If we compare this situation to the landscape of all expressions (Figures B.1 and B.2), we observe the same, with a single exception. The category of 14 grammatical parses dominates all other parse counts, simply because of the high frequency of single letter expressions. A single letter is considered in the "Any" grammatical category, which indeed produces 14 parses in total. As it is safe to discount these expressions as trivial, we can conclude with a very satisfactory overall performance in all categories.

7.4 Expression Length

Next, let us reexamine these results in the context of expression length. We consider the number of lexemes[1] building up an expression to be an interesting property to examine, as longer expressions provide more context for disambiguation.

We again present four cases that capture the expression lengths (or "lexeme counts"), for both test and train sets, in Figures B.5, B.6, B.7 and B.8.

We immediately spot the normal distribution over odd-length expressions in Figure 7.2, over all unique baseline expressions, a trace of which is also visible in the cumulative baseline expressions (Figure B.5). This in itself is a very interesting observation, exhibited

[1]Note that, from a grammatical perspective, we use the words tokens, symbols and lexemes interchangibly, with the meaning of lexical atoms in mathematical expressions.

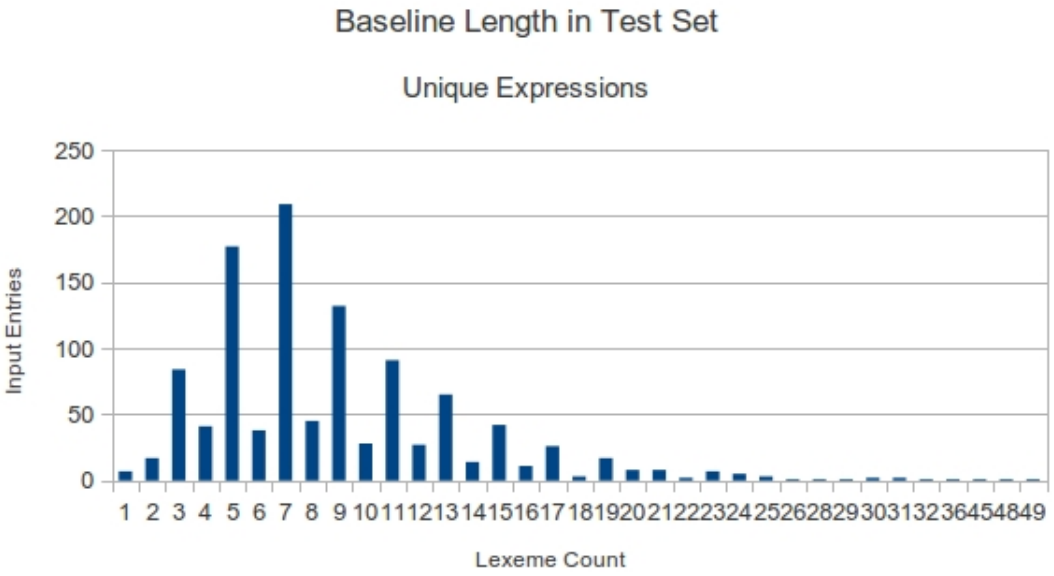


Figure 7.2: Lexeme Count over Unique Baseline Expressions in Test Set

by both document sets.[2] In the cumulative plots, we see an overwhelming number of single-letter expressions, while the unique fragment plots still look inconclusive.

We now have a reason to expect a certain preference to diverse expressions of about 7 lexemes, as the peak of a normal distribution. The diversity is much less on both extremes, as there are only that many symbols that authors can use stand-alone[3] and not too many sizeable expressions (20 lexemes or more) in our collections.

However, this gives us no grounds to expect some specific behaviour of how the number of parses varies with the number of lexemes. We explore these correlations in Figures B.9, B.10, B.11, B.12

It is surprising that, again after discounting the case of a single symbol, all plots agree on a roughly homogeneous spread of the discovered parses, as the expression length grows. This is a strong indication that we have indeed succeeded in recognizing a sufficient portion of the disambiguation cues provided by the authors, as to decrease the degree of ambiguity irrespective of the size of the expression. Again, this is a satisfactory result, with the average parses being roughly around four per expression.

[2]The even-length expressions are around 20% of the overall expressions. The author suspects this is due to the frequent use of common relational expressions, such as (in)equalities.

[3]We observe only 7 or 8 symbols (letters, numbers and operators) used independently as complete baseline expressions, and about 15 symbols used as stand-alone fragments.

7.5 Error Analysis

Having reviewed the (roughly) 100 overall examples that were not successfully parsed by the grammar, the author notes on the following classes of errors that occurred:

1. ungrammatical inputs and typos [$\approx$ 40 *inputs*]:
   (a) trailing infix operators 1/2/,
   (b) unbalanced fences,
   (c) using dot instead of comma in sequences such as with $j.k$
   (d) misuse of mathematical operators as narrative symbols (e.g. using the infix operator $\bullet$ as an itemization bullet),
   (e) trailing sequence operators, e.g. "$1, 2,$",
2. normalization problems of inputs [$\approx$ 15 *inputs*]: some elements of the XML representation are still not properly treated in the conversion to a CCG input (e.g. XMWrap and XMDual).
3. concatenation for operators with ambiguous fixity [$\approx$ 15 *inputs*]: e.g. a "bra" following a fenced expression is currently ungrammatical, as no concatenation is present, e.g. in $(A2)|N1\rangle$. This problem is shared by fences that can be both opening and closing, such as bar, but also [and] in groups and French intervals. A more general treatment should be sought after.
4. out of scope inputs [$\approx$ 10 *inputs*]: about piecewise functions were treated as grammatical failures
5. lexer mistakes [$\approx$ 10 *inputs*] – in some cases $\mathbb{R}$ is treated as big operator instead of term.
6. domain-specific notations [$\approx$ 5 *inputs*] – in the Computational Complexity entry in our test set, the notation $[i..j]$ is introduced and used throughout the paper, yet is something that our grammar can not be prepared for. This is the exact motivation for an adaptive approach to a full solution, which we discussed in Section 5.1.

The majority of ungrammatical cases in which parsing failed, were due to two types of author misdemeanor – classic typos and, secondly, presentational misuse of the math modality in LaTeX. We have already outlined possible ways of addressing these issues (c.f. Section 5.1.3). What is ceratin is that the treatment of ungrammatical inputs must indeed remain outside of the scope of our grammar.

Nevertheless, there are indeed phenomena which our grammar still needs to address. The interplay of operators and fences with ambiguous fixity, and the challenge of properly placing concatenation operators in these cases, is a primary concern and is an instance which the author would consider immediate future work.

A minority of the failures were due to author-defined notations, which would require an adaptive treatment. While this reconfirms our intuition of a bigger challenge lying ahead, it also confirms our expectation to be able to parse a majority of math expressions, on a purely syntactic and domain-independent basis.

7.6 Summary

We have shown that our grammar has achieved a wide coverage with well-behaved degrees of ambiguity in its output parses. But more importantly, we have decisively confirmed the intuition that the structure of mathematical symbolism is retrievable on the grounds of syntax and internal expression context, in a big majority of real-world expressions.

This is a strong result. Our comparisons between unique and cumulative parses, as well as the average parses per expression length, indicate that the presented grammar is uniformly well-behaved on our document collections.

The author sees a three-fold outlook to these findings. First, a reliable mechanism is needed in order to deal with ungrammatical inputs, observing that negligence invariably occurs in real world discourse, leading to typos or modality misuse. Second, we have to improve our grammar coverage in the directions of operators with transfix or ambiguous fixity, as well as to incorporate domain knowledge into the grammar rules. Lastly, in the long term, there is an evident need of an adaptive approach which can adopt new notations as the author introduces them, a problem which the author believes to be AI-hard and is tightly connected to understanding both the symbolic and language semiotic of scientific texts.

Chapter 8

Discussion

Armed with our new understanding of the landscape of mathematical symbolism, let us revisit some of the relevant related work that we already outlined in Chapter 2, in order to compare and contrast it with our findings.

8.1 The LaTeXML Grammar

The LaTeXML grammar is designed and developed on two large document collections (arXMLiv and DLMF [DLM]) and successfully parses all DLMF sources, together with at least 34,000 scientific documents from arXMLiv.

However, it has some known deficiencies which we will briefly enumerate:

1. based on Perl's RecDescent module, essentially a context-free grammar without support for multiple parses.

2. heuristic inference of the presence of concatenation with nontrivial semantics (e.g. invisible operators).

3. when inferred to be non-trivial, concatenation is defaulted to invisible times.

4. designed in a knowledge-rich approach, so that a specific construct (for example a group constructor) would be parsed only if a specific rule exists for it. Incidentally, currently (almost) all operator symbols of arithmetic and logic have supporting grammatical derivations that only support a single behaviour, typically the one exhibited in basic arithmetic.

On the other hand, the LaTeXML grammar has already embraced the challenges of multi-line expressions and two-dimensional constructs, making it the most advanced tool that is fully aware of the representation of mathematical expressions.

The examples in the previous chapters have indicated that this approach lacks the needed complexity and flexibility in order to scale to arbitrary mathematics. This is

essentially not surprising, as the original intention, and the current state, of the LaTeXML grammar is to provide a reliable mechanism for the construction of layout trees, with just a first outlook to the creation of operator trees. The author would pursue a redesign and improvement of the LaTeXML grammar as a major direction of future work. This is not to state an intention to throw away the current progress established in this thesis – one possibility is to include OpenCCG as a component in LaTeXML. Another would be to find an alternative grammar toolbox that is suitable for capturing the phenomena discussed in Chapter 4, but is a better fit to LaTeXML than OpenCCG is. The author would be interested in investigating the various directions, as natively providing LaTeXML with a mechanism for generating operator trees, would be of high impact not only for converting the available large scale corpora, but also for various projects in the MKM community.

8.2 The Language of Mathematics

It is illuminating to pay special attention to Ganesalingam's PhD thesis, "The Language of Mathematics" [Gan09], which is concerned with the full spectrum of challenges on the way to a computational understanding of mathematical language.

Ganesalingam's thesis plunges into the full depth of the problem, considering the linguistic and semantic challenges that arise on the way, but staying relatively narrow in the breadth of mathematical language actually considered. In opposition, the current work takes a data-driven broad approach to analyzing but a single stage of that process, paying much closer attention to the arising phenomena and necessary supporting infrastructure.

On an abstract level, the author shares Ganesalingam's intuition of an "adaptive" approach to understanding mathematical language, namely an approach that is flexible in its supported lexicon and grammar of mathematical symbolism and can dynamically adapt them during runtime, i.e. during the interpretation of a given document, similarly to the way a human reader would. However, the actual foundations on which this idea is built on seems to be in a rather infant stage in Ganesalingam's work.

For example, it is probably mistaken to mix the static properties of mathematical knowledge, as entailed by its document carrier, with the actual interpretation of such a document by a reader. The error is visible in Ganesalingam's motivation for incorporating a notion of "time" for document understanding. Instead, it is of greater clarity and power to operate in two separate domains. For the modeling of mathematical knowledge, OMDoc [Koh10] is a strong candidate for an ontological model of mathematical discourse, while framing [KK09], built on top of an MMT core [RK11], can give an accurate account of the incremental understanding, as well as the multiple views over a given mathematical concept, by simply accounting for the reference theories from which one views, or tries to understand, an object. From such a perspective, Ganesalingam's choice of non-extensional typing becomes obsolete, and the process of understanding is reduced to incrementally building a logical model of the interpreted discourse, compliant with an underlying knowledge base in MMT, which would also dynamically evolve along the way. However, this larger vision is out of the scope of the current thesis and we make these remarks only in

order to clarify the reference point of the work outlined further on.

Our stance is clear, having seen that we can achieve wide coverage parsing of mathematical expressions with domain-independent syntactic methods, an intuition that is clearly not shared by Ganesalingam's work. In this thesis we have allured to an envisioned two stage workflow, starting with a (mostly) syntactic process to establish expression structure and subsequently continuing with semantic analysis to establish the concrete denotations, with adaptivity ensured in both stages. Nevertheless, there are a lot of common parts in the two visions, as well as multiple shared observations, which we will survey below.

It is useful to enumerate a number of analyses in "The Language of Mathematics" that were inspiring and at times eye-opening:

1. a distinction between formal and informal textual components
2. a demonstration of how textual and symbolic fragments are inter-dependent for disambiguation.
3. an in-depth motivation of the need of syntactic types (i.e. a type system for parsing), which the author had previously taken for granted.
4. a justification of the orthogonality between types and properties such as precedence or associativity of infix operators.
5. examples of how formulas act as noun phrase(NP) or sentence(S) parts of speech when interacting with textual fragments. Nevertheless, that classification is incomplete, as formulas can also stand for numeral adjectives:

   "If you take $2n + 1$ *apples..."*
6. a distinction between intensional and extensional behaviors of the syntax derivations, when dealing with syntactic subsumption [Gan09, page 109]. The example is $\sin^{-1}$, which is to be interpreted as arcsin and (intentional), as opposed to the inverse of a function (extensional).
7. Useful examples of word sense ambiguity in textual mathematics:
   - polysemy (prime ideals, prime numbers)
   - homonymy (normal subgroups, normal polynomials).

There are also several occasions where the author's intuition diverges from that of Ganesalingam:

1. the use of classical context-free grammars. Ganesalingam's justification is that "textual mathematical language" is a context-free subset of language. However, while I have not found any counter-examples to this statement, I believe the jury is still out. In symbolic expressions, most of the known context-free phenomena, such as verb phrase coordination, do not occur, possibly because the economy achieved is minimal

(the following NP is usually a single symbol) and the expressions become too difficult to read. For example, "$x \in, y \notin \mathbb{N}$" is not a grammatical expression in matehmatics. Insetad, authors tend to construct context-free argument sequences, such as $x, y \in \mathbb{N}$ or $x \in \mathbb{N}, \mathbb{Z}$. However, the author is yet to encounter a systematic study of how complex operators interrelate, e.g. when chaining transfix operators or quantifiers, especially in non-standard notations, which are good suspects for context-sensitive behaviour.

2. as mentioned above, the "special notion of time" in mathematics, would be better modeled in terms of theory imports and definition scopes, as provided by the MMT framework and the OMDoc language. Ganesalingam's justification seems contradictory, as a well-designed demonstration of mathematics being a timeless language was also presented.

3. the analysis of the blocks within mathematical discourse remind of a poor-man's version of the OMDoc ontology, which is preferable because it is more exhaustive and systematic.

4. there is a contradiction in the views about what types should represent, during the syntactic parsing. There are fundamentalist views about always using ZF set theory as a foundation that should be incrementally expanded via adaptivity, but also claims that Number or even Reals should be syntactic types, making some sets more "special" than others. On yet another side, "adaptivity" itself is supposed to introduce the Reals as a syntactic type, which is specified in an incomplete way prone to lose all structural information. This could be interpreted as another sign of confusion arising from the merge between using a framework for modeling mathematical knowledge and having a separate theory of interpretation.

5. the analysis of concatenation and operators misunderstands the problem as attributing their syntactic roles to the actual semantic types of the lexemes. Instead, as we have shown in Section 4.1.3, the syntactic roles of operators and concatenation can be disambiguated fully by the internal context of an expression. This allows for a more natural grammar design, as we demonstrate in Section 5.2.

Chapter 9

Representation

In Chapter 2 we described the use of the OpenMath [BCC+04] and Content MathML [ABC+10] standards for making the content of mathematical expressions transparent to computer applications. We did not mention, however, that both standards assume that the objects they represent are fully disambiguated and semantically well-specified. As we discussed in Chapter 3, this assumption is too strong for computational agents, which can not properly emulate the background knowledge, context and interpretation of a human reader.

To this end, we design an extension to these formats, by a minimal set of constructs, so that one could capture the content of a potentially ambiguous expression, via serializing its CDLF (see Section 5.2.1). Ideally, one would limit such extensions to the introduction of a single disjunction element, as well as a couple of generic elements for underspecified tokens and structural operations (e.g. underspecified script operators).

Besides the theoretical adequacy, there are also practical implications that need to be taken into consideration. As a fully expanded ambiguous formula would have an exponentially larger size and exponentially slower traversal time than its underspecified form, it becomes clear that one needs to utilize compacted trees of such expressions. As an example, compactification allows for recursive, context-driven disambiguation while reducing both memory and complexity of the inference process via localizing the pruning of irrelevant subtrees.

Another valuable observation is that different underspecified trees might represent the same content object, e.g. when commutativity and associativity hold for an operator. It is hard to decide on semantic identity of symbols (and structure) without further information on domain practices and document context. These difficulties are important when considering further reducing representation size via cross-referencing identical subtrees, and further on when designing applications that build on this representation, such as mathematics search engines. While it is good to eliminate redundancy, one must make a clear distinction of whether an object is underspecified and claim it redundant only to objects not only sharing the same sign composition, but also the exact degree of underspecification.

This thesis provides a generic minimal extension to XML standards for content math-

ematics, and an implementation of API library support for various common tasks, such as packing and unpacking of expressions, auto cross-referencing and dereferencing, as well as translations down to the current versions of OpenMath and Content MathML, by enumeration of all possible meanings. The software is available under the GPL free open-source license, and can be checked out together with the LaMaPUn library [Gin11].

To illustrate our methods, we have enclosed a simple running example in Appendix C. The appendix follows the conversion and analysis of the expression "$x, y \in S$" throughout the LaMaPUn pipeline, presenting the various representations in the process.

9.1 CDLF XML Elements

In this section we present a minimal set of elements, used to described compact disjunctive logical forms, originally created as parse forests for mathematical expressions.

First, let us justify why our extensions do not make use of the already existing extension mechanisms within Content MathML and OpenMath, known as "Content Dictionaries". This is the case, as an extension within the content standards would entail treating the disjunction of subtrees and the underspecification of symbols as mathematical objects. However, they are one meta level higher, as the captured properties are those of the structure of XML forests, as opposed of the properties of the expression itself. To avoid confusion, and have a clear separation of concerns, we consider the CDLF elements as external to the mathematical objects.

In other words, we adopt a convention of using underspecified elements as distinct mixins within the classic standards for content mathematics. This is simple to achieve, as all we require is a separate namespace to host our CDLF elements, and a schema that permits its use. To keep things simple, we will base all our examples on Content MathML 3, which has recently converged with OpenMath.

We introduce two elements and explain a generic treatment for underspecified operations:

9.1.1 The Set Element

A disjunctive logical form contains, by definition, disjunctions to represent mutually exclusive parse options. In the OpenCCG logical forms, a disjunction is represented as a simple enumeration of the disjuncts, bound together by a common *lf* parent element.

Similarly, we introduce a *cdlf:set* element, which will act as a binding parent for mutually exclusive parse trees, which in turn typically have an *m:apply* root, or a single leaf[1].

[1]In our underspecified setting we are yet to address variable binders, hence we currently only consider applications.

9.1.2 The Token Element

While disjunctions are a needed prerequisite, a compact disjunctive logical form also needs to capture the underspecification of its symbols, when needed. In section 3 we discussed various kinds of ambiguity and the need of their resolution in a follow-up semantic analysis stage. In order for such resolution to be possible, we need to be able to represent tokens on the merit of their structural, and not semantic, properties. For example, having a binary operator "+" allows us to proceed with further analysis as to whether it is an arithmetic or set operator. To this extent, we should not represent "+" as a disambiguated *m:csymbol* in Content MathML, but as a newly introduced *cdlf:token* element, specifying its structural properties, e.g.

```
<cdlf:token cat="term\*term/^term" lex="inop1" pos="INF-OP">+</token>
```

We propose a *cdlf:token* element, carrying over the category, lexeme, part-of-speech and presentation of a parsed grammar lexeme. One can utilize this representation for underspecified search and analysis, eventually disambiguating it into the standard *m:csymbol* semantic symbols of Content MathML. The default target of all parsed lexemes by the project's extended CCG architecture are namely such *cdlf:token* elements.

9.1.3 Underspecified Operators

To briefly remark on underspecified operators, such as scripted operators (which we treat compositionally, see Section 5.1.4), we need only state that they are well-captured via the generic *cdlf:token* elements. This is the case, as each operator application is represented as a tree with an *m:apply* root and an underspecified operator as a first child, residing exactly in a *cdlf:token* element. The second child of the application, in our script example, would be the subtree corresponding to the parses of the subexpression residing in the sub- or super-script.

Hence, we consider these two generic elements as sufficient to fully represent underspecification and structural ambiguity in math symbolism. In our running example, one can observe how the logical forms are captured by these symbols by examining the XML snippet enclosed in Appendix C.5.

9.2 Equivalence of XML Trees

We base our approach to reducing redundancy in the XML representation of parse forests on a flexible notion of equivalence of XML trees. Empowered with the equivalence classes of the subtrees in a CDLF tree, it becomes trivial to enable sharing (or cross-referencing) of equivalent components. Additionally, the equivalence notion provides solid footing for a graph-theoretic view on compacting parse forests.

In order to scale between the representations of different parsers, the author has made the criteria for equivalence parametric in a range of markers. As we are approaching the problem from a representational perspective, the markers can be selected from a subset

of the XML features available, namely attributes, namespaces, node names and textual content. We have also built in basic awareness of element semantics – e.g. resolving element references.

While the notion of equivalence is flexible on the node level, the consideration of children has a fixed interpretation. Two equivalent subtrees must have equivalent root nodes, the same number of children, and each child of the first tree should match a respective child in the second subtree in the same position and of the same equivalence class.

It is already evident that this enables the incremental labeling with equivalence classes starting from the leaves and proceeding up level by level. We exploit this with a dynamic programming algorithm that creates all classes in linear time of the size of the tree ($b \times d \times C$, where b stands for tree breadth, d for tree depth and C is the lookup constant for Perl's hash tables).

The implementation and API of the equivalence algorithm can be found at the corresponding module of the LaMaPUn library [Gin11] – "LaMaPUn::Util::XML::Equivalence".

To illustrate, consider the equivalence graph of the XML representation in Appendix C.6. To make this more accessible for untrained eyes, we extract the equivalence tree of each of the five possible logical forms in Table 9.1. Note that we omit the root *cdlf:set* node which acts as the top level disjunction wrapped around the parse forest.

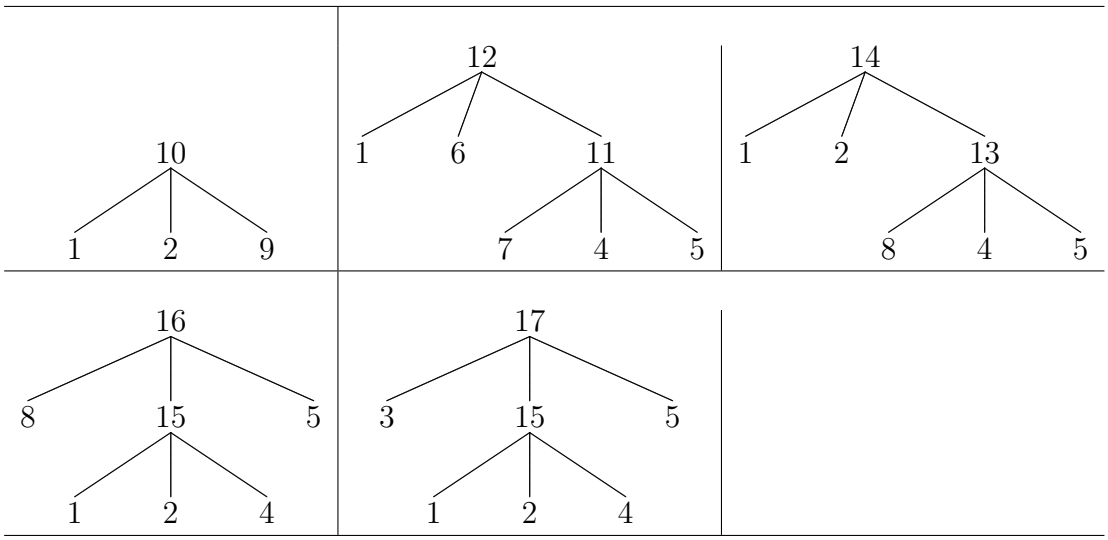


Table 9.1: Equivalence tree of Appendix C.6

9.3 Compacting XML Forests

As in the case of OpenCCG, the author expects a forest of possible parses with none or little sharing of elements[2]. These parses, if kept as-is, tend to be largely redundant. For example, an expression of 20 lexemes could have only two parses, differing by a single leaf or a terminal subtree – a case in which a compact form would reduce the size of this forest by two. We can see that when structural ambiguity increases, the packing can be denser (i.e. multiple trees can be packed together in a single disjunctive tree), which makes the saving ratio larger.

The greater utility of compact forms is not in the size of their representation, however. Tools for further analysis or applications could immediately benefit from the compact forms, by efficiently pruning unlikely trees or smartly indexing disjunctive blocks. We consider this an important benefit that should be pursued.

Turning to the construction of such forms, let us consider forests enriched with equivalence class labels, as introduced by the previous section. Knowing the equivalences allows us to escape our representational XML perspective, and to return the problem to graph theory.

The author has established that the compacting problem is indeed an optimization problem for rewriting a directed acyclic graph (DAG), formed by the equivalence classes and the parenthood relations of the parse forest, into a minimal graph subject to several constraints.[3]

Back to our running example, we present a CDLF XML representation in Appendix C.7. Again, for the reader's convenience, we also present the extracted equivalence graph (omitting the top level disjunction) in Table 9.2. We have also explicitly added a label "SET" to the nodes with equivalence classes representing a disjunction, for further clarity.

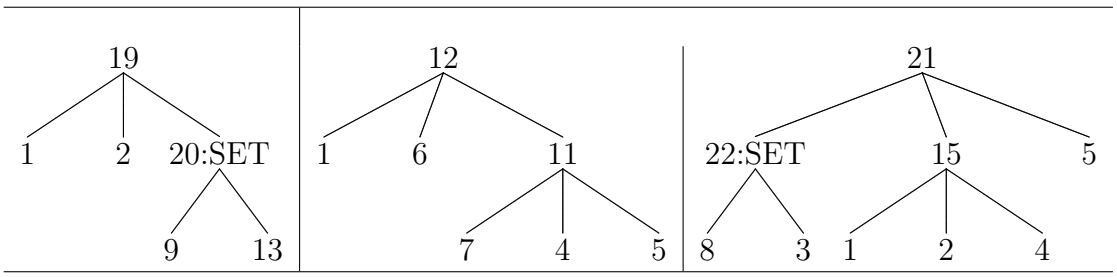


Table 9.2: Equivalence tree of Appendix C.7

An important observation rests in recognizing the different trade-offs in compacting. One could optimize towards a least number of final trees in the forest, or alternatively

[2]The OpenCCG library has a very badly maintained approach to sharing of its output parses that manages to cause more harm than benefit. We preprocess their XML into an explicit forest without any sharing before proceeding to compacting.

[3]The author is developing an algorithm that detects optimal cliques for compacting, based on a topological search for bipartite graphs that satisfy a linear system of equations on their in/out-degrees.

towards a maximal proximity between the compacted parses, in order to achieve a highest degree of compression. The two goals are conflicting and the author is not certain about the different benefits they entail.

As this direction of our work has not been fully completed, we will conclude with only making the above remarks and point the readers to an ongoing implementation effort (currently using a greedy heuristic) at the module "LaMaPUn::Util::XML::SharedPacked". The same module also hosts the implementation of the sharing algorithms, described in the following section.

There has been a large body of related work in creating packed representations, however from different perspectives, such as parser development in Machine Translation [Hua08] and decreasing redundancy when generating XML storage structures [ME06].

9.4 Sharing of XML Trees

The idea behind sharing of trees, is to replace all equivalent trees, but one, with a reference element. We think of a primary tree and references to it, which decreases redundancy and size of the representation. Also, as the equivalence labels are temporary and will not be available in the final representation, the shared subtrees preserve the notion of equivalence, which is useful for disambiguating the input in various end-user applications or further analysis stages.

Using the established notion of tree equivalence from Section 9.2, it is straightforward to enable sharing of identical subtrees, on the basis of identical equivalence class labels. This process is so immediate, that we see no need of more in depth discussion to this extent.

There is an important reason why sharing should be postponed until after the compacting stage. Namely, the compacting procedure is kept simple, as it does not need to dereference, or generally take into account, any reference information. Another problem would be a hit to efficiency, as the reorganization of disjuncts during compacting induces new equivalence classes, which should also be eventually shared. Hence, we position sharing as the last step of the representation pipeline.

Another instance of sharing would be cross-referencing between the logical forms and a parallel presentation format containing the layout tree of the expression, which is considered future work to this thesis that would have high-impact for applications in the domain of "Active Documents", such as definition lookup or a clipboard for mathematical formulas.

Let us conclude by remarking that a sharing-enabled CDLF representation of the running example is presented in Appendix C.8.

Chapter 10

Conclusion and Future Work

The project described in this thesis attempts to make the next big move towards the evolution of social knowledge, applicable to the digital era. The author envisions this change to occur through the inclusion of computationally transparent understanding of content, supported by special semantic formats. The intention is to open the road to added value services, exemplified by enabling content math retrieval on legacy corpora. We have restricted ourselves to the symbolic semiotic of mathematical discourse, as it has both the most mature representation efforts in the state of art and also represents its content via a sign-system that is closest to being fully formal, and hence machine understandable.

The suggested approach considers two separate subproblems – structure recovery and representation, rooted at two different scientific subdomains, namely Computational Linguistics and Mathematical Knowledge Management.

To this extent, we have surveyed in detail the phenomena of ambiguity and underspecification (c.f. Chapter 3), as well as the overall landscape of structural phenomena in mathematical symbolism (c.f. Chapter 4), gaining a lot of insight into the complexity and inter-dependencies that occur in real world texts. We have developed a first version of a Combinatory Categorial grammar for mathematical symbolism, which has already been evaluated to show high quality and coverage, sufficient to achieve a high impact in mathematics retrieval, but also to set a sound foundation for future work on disambiguation of mathematical expressions. We have seen a clear separation between syntactic and semantic concerns in the process of extracting the operator trees of mathematical expressions, which both gives hope to broad coverage tools such as LaTeXML and even more so could enhance the coverage of controlled natural language tools, such as the Naproche system, among others.

To put a strong result to practice, the thesis develops a generic approach to support dealing with compact disjunctive logical forms (CDLF) from a representational perspective. We have grounded our understanding on a flexible notion of equivalence, allowing our treatment to scale to arbitrary XML syntax of the forms, and provide the methods to introduce and remove sharing and compactness on a CDLF input. We have provided a complete off-the-shelf toolbox that supports these methods, contributing a high value

practical result to this thesis component.

The implementation of the various components has been developed within the LaMaPUn framework, specially designed for dealing with scientific publications, and more generically documents authored using the TEX/LATEX typesetting systems. All our results are hence available under permissive open-source licenses and will be continuously developed and maintained in the future.

10.1 Future Work: Applications

We have built and tested an analysis framework and clarified the different components. As we consider realizing its potential, we shift our focus on demonstrating the utility and impact of our work.

It is a clear short-term goal to integrate the result of the CDLF construction process with an array of content-based applications. Notably, we intend to enrich the arXMLiv corpus with the CDLF format for a two-fold purpose. On one hand, we put in place a fundament that should be used as a first step for further disambiguation of mathematical expressions by the LaMaPUn framework [GJA+09]. On the other hand, we provide semantic markup of formulas, which can be then used for providing semantic services. Some of the tools we directly consider to be of high impact are:

10.1.1 Content-based Mathematics Retrieval

We plan on integrating the CDLF representation with the index of MathWebSearch [KŞ06, PK11], so that we achieve a many-to-many retrieval process on both queries and indexed data over underspecified expressions. As the prerequisite of enriching a large-scale corpus with CDLF forms is in sight, namely arXMLiv, the new challenges are to provide the right user workflow and capabilities for creating CDLF expressions.

A typical workflow we envision is of a user writing a formula in slightly enhanced LATEX, capable of explicitly designating query variables, which then gets converted to a CDLF form with explicitly authored placeholders for said variables. The next internal step is to extract a MWS query from this CDLF operator tree. In the short term, this can be done by unpacking the possible meanings of the disjunctive logical form, sending a set of queries, each for a formula with unambiguous semantics, and returning back to the user a smartly aggregated result set, consisting of URI references to the matched formulas.

In the mid-term there are two directions in which to improve this approach. The first is to generalize the input capabilities of MWS to allow queries that represent forests, each internally handled as a small database index. This is an enhancement that the indexing method of MathWebSearch, called Substitution Tree Indexing, already supports and is largely an implementational task.

The second direction of work is to provide a disambiguation interface that utilizes cross-references between the presentation and content objects within a CDLF expression with parallel Presentation MathML annotations, which deserves to be discussed in its own right

and we do so in Section 10.1.2. In brief, the UI is to suggest subexpressions that need further specification or structure that needs to be disambiguated, and guide the user into painlessly supplying the needed clarifications. This is a relevant add-on, as searching for a well-specified expression with unambiguous semantics guarantees a higher relevance of the matched results. Alternatively, disambiguating expressions within the corpus entries, provides a higher value to the respective document as it promises higher result relevance when a related formula is queried for.

As an outlook to future technologies, this integration will also enable the large-scale operation of novel search procedures, such as Applicable Theorem Search (ATS) [Anc09, PK11]. ATS attempts retrieval of expression in their theory context, in order to enable more refined user needs, such as finding the definition of an object, searching for axioms related to a given set, etc. As ATS also bridges the modalities of natural language and mathematical symbolism, we see it as a relevant application for synergies with future work related to the analysis component of this thesis.

Even without these enhancements, enabling queries over CDLF expressions is extremely exciting, as it enables large-scale content mathematics search for the casual user, remaining in their comfort zone of authoring mathematics in LaTeX. One must admit that the perspective of having a mathematical resource of over ten megaformuli under the tips of one's fingers, would be a very significant outcome, which would be a first of its kind success in such magnitude.

10.1.2 Interactive Authoring of Formulas

Being part of an easy to reuse framework, the extraction process can be incorporated in various plug-ins for user assistance. We are primarily interested in developing an interface for disambiguating expressions on input, when authored in presentational formats such as LaTeX. A direct integration is envisioned into the GUI component of the MathWebSearch search engine.

Interactively polling the user for further specifying subexpressions or resolving ambiguities, would be a strong aid in the semantic publishing of documents in platforms such as the Planetary system [DGKC10]. This is the case, as interactively disambiguating the expressions achieves a two-fold purpose. First, it avoids typos and invalid expressions, which authors tend to create when they lack an incentive to explicitly avoid them. Second, documents with well-authored semantic expressions will have higher add-on value of being published, as their mathematical content would be easily found, discussed and referenced by other users.

As a parallel side benefit, the process of user-driven disambiguation could be recorded in order to build a database of CDLF expressions, together with their disambiguation in context, which could be used as a gold standard for the general challenge of disambiguating mathematical expressions. The creation of this database is trivial and for example requires but a simple export of the index of the MathWebSearch installation.

In order to build the authoring mechanism towards this disambiguation, a first prerequisite is a service with the capacity for on-the-fly conversion of formulas. The author has

already created such a service, namely a daemon for the LaTeXML software stack [GSK11]. Once coupled with the CCG grammar, this tool would solve the efficiency and usability requirements.

A last missing component beyond the authoring process would be an interface for the disambiguation process. We are aware of initial work in this direction, developed by Mihai Grigore in Saarland University [Gri] and Catalin David at Jacobs University Bremen [GJA+09].

10.1.3 A Clipboard for Mathematical Formulas

A service that the MKM community has long asked for is a context-aware clipboard, capable of providing an adequate representation based on the service it is active within. A specification for its capabilities has already been chartered by the MathML working group [ABC+10, Chapter 6.3 Transferring MathML], but an implementation is yet to be created.

As an example, a clipboard service should be able to allow the user to seamlessly transition between copying a formula in their browser (presentation), pasting it in their authoring environment of choice (source), and after modification copy-pasting again into a Computer Algebra System (content).

10.2 Future Work: Analysis

We see different directions of work on improving our analytic contributions, driven by distinct purposes:

10.2.1 Improvements in Scale

As we have indicated in Section 8.1, one of our long-term desires is to couple a form of our grammatical analysis natively with the LaTeXML system, in order to achieve out-of-the-box creation of underspecified content expressions from arbitrary LaTeX documents. To that extent, we are also determined to accommodate all grammatical failures that we have uncovered in our grammar during our evaluation (c.f. Section 7.5), while in parallel increase its coverage by collecting data from larger subsets of the arXMLiv corpus. The build system of the arXMLiv project [SKG+10] could be easily utilized to harvest and classify the parsing results over the entire document collection of over 600,000. Actually, this is already being done for the parses of LaTeXML's grammar, however these statistics have not yet been systematically analyzed.

10.2.2 Improvements in Modeling

The improvements in scale must necessarily be based on ever better modeling of the phenomena in mathematical symbolism. The author is already aware of inaccuracies in the

current treatment of concatenation, and is yet to provide support for domain-specific notations, such as transfix operators.

10.2.3 Improvements in Integration

Coming to the hardest problem, there is only so much that an accurate model of the syntax of mathematical symbols can achieve.

In order to achieve truly human coverage and precision, there is no way around ultimatively using contextual information about the domain, document-level statements and theories, as well as to allow a certain degree of adaptivity to inline syntactic conventions. A short-term goal in this direction would be integration with some of the tools that have been developed by Magdalena Wolska and Mihai Grigore in Saarland, such as [GWK09, WG10, WGK11].

10.3 Future Work: Representation

When it comes to the representation side of the thesis is concerned, the author feels comfortable with the current solution for representing CDLF forests, as well as with the sharing and equivalence toolboxes. As far as compacting is concerned, there is good initial work on understanding the challenge from a graph theoretic perspective, a full algorithm towards which should be ultimately pursued. However, the need for an elaborate treatment of compacting is yet to be demonstrated, as we have seen that highly ambiguous parses are scarce, if at all present.

Bibliography

[ABC+10] Ron Ausbrooks, Stephen Buswell, David Carlisle, Giorgi Chavchanidze, Stéphane Dalmas, Stan Devitt, Angel Diaz, Sam Dooley, Roger Hunter, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Paul Libbrecht, Bruce Miller, Robert Miner, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 3.0. W3C Recommendation, World Wide Web Consortium (W3C), 2010.

[ABMH] Ben Adida, Mark Birbeck, Shane McCarron, and Ivan Herman. RDFa core 1.1. W3C Working Draft, World Wide Web Consortium (W3C).

[ABT04] Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec, editors. *Mathematical Knowledge Management, MKM'04*, number 3119 in LNAI. Springer Verlag, 2004.

[Anc09] Ştefan Anca. Natural language and mathematics processing for applicable theorem search. Master's thesis, Jacobs University Bremen, 2009.

[ArX] `arxiv.org` e-Print archive. web page at `http://www.arxiv.org`.

[AS04] Andrea Asperti and Matteo Selmi. Efficient retrieval of mathematical statements. In Asperti et al. [ABT04], pages 1–4.

[AvH04] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.

[Bat08] John Bateman. *Multimodality and Genre: A Foundation for the Systematic Analysis of Multimodal Documents*. Palgrave Macmillan, 2008.

[BB05] Mark Buckley and Christoph Benzmüller. System Description: A Dialogue Manager supporting Natural Language Tutorial Dialogue on Proofs. In David Aspinall and Christoph Lüth, editors, *Proceedings of the ETAPS Satellite Workshop on User Interfaces for Theorem Provers (UITP)*, pages 40–67, Edinburgh, Scotland, 2005.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaëtano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The OpenMath Society, 2004.

[BMS03] Christoph Benzmüller, Andreas Meier, and Volker Sorge. Bridging theorem proving and mathematical knowledge retrieval. In *Festschrift in Honour of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer, 2003. To appear.

[Bos98] Cascading style sheets, level 2; css2 specification. W3C recommendation, World Wide Web Consortium (W3C), 1998.

[Caj93] Florian Cajori. *A History of Mathematical Notations*. Courier Dover Publications, 1993. Originally published in 1929.

[CCG09] OpenCCG Library. Project home page at `http://openccg.sourceforge.net/`, seen November 2009.

[CKS11] Marcos Cramer, Peter Koepke, and Bernhard Schröder. Parsing and disambiguation of symbolic mathematics in the naproche system. pages 180–195, 2011.

[CZ04] Claudio Sacerdoti Coen and Stefano Zacchiroli. Efficient ambiguous parsing of mathematical formulae. In Asperti et al. [ABT04], pages 347–362.

[DGKC10] Catalin David, Deyan Ginev, Michael Kohlhase, and Joe Corneli. eMath 3.0: Building blocks for a social and semantic web for online mathematics & ELearning. In Ion Mierlus-Mazilu, editor, *1st International Workshop on Mathematics and ICT: Education, Research and Applications*, 2010.

[DLM] Digital Library of Mathematical Functions.

[DLR10] Catalin David, Christoph Lange, and Florian Rabe. Interactive documents as interfaces to computer algebra systems: JOBAD and Wolfram—Alpha. In David Delahaye and Renaud Rioboo, editors, *CALCULEMUS (Emerging Trends)*, pages 13–30. Centre d'Étude et de Recherche en Informatique du CNAM (Cédric), 2010.

[EuD] EuDML – the European digital mathematics library.

[For07] Help: Displaying a Formula. `http://en.wikipedia.org/w/index.php?title=Help:Displaying_a_formula&oldid=444038561`, 2001–2007. seen August 2011.

[Gan09] Mohan Ganesalingam. *The Language of Mathematics*. PhD thesis, Cambridge University, 2009.

[Gin11] Deyan Ginev. LaMaPUn OOPerl Library. Development status at `https://trac.kwarc.info/lamapun/ticket/21`, seen August 2011.

[GJA+09] Deyan Ginev, Constantin Jucovschi, Stefan Anca, Mihai Grigore, Catalin David, and Michael Kohlhase. An architecture for linguistic and semantic analysis on the arXMLiv corpus. In *Applications of Semantic Technologies (AST) Workshop at Informatik 2009*, 2009.

[Gri] Mihai Grigore. Personal communication to D. Ginev. February 23 2011.

[Gri10] Mihai Grigore. Knowledge-poor Interpretation of Mathematical Expressions in Context. Master thesis, Jacobs University Bremen, Bremen, Germany, August 2010.

[GSK11] Deyan Ginev, Heinrich Stamerjohanns, and Michael Kohlhase. The LaTeXML daemon: Editable math on the collaborative web. In James Davenport, William Farmer, Florian Rabe, and Josef Urban, editors, *Intelligent Computer Mathematics*, number 6824 in LNAI. Springer Verlag, 2011. in press.

[GWK09] Mihai Grigore, Magdalena Wolska, and Michael Kohlhase. Towards context-based disambiguation of mathematical expressions. In Masakazu Suzuki, Hoon Hong, Hirokazu Anai, Chee Yap, Yousuke Sato, and Hiroshi Yoshida, editors, *The Joint Conference of ASCM 2009 and MACIS 2009: Asian Symposium on Computer Mathematics and Mathematical Aspects of Computer and Information Sciences*, volume 22 of *COE Lecture Notes*, pages 262–271, Fukuoka, Japan, December 2009. Faculty of Mathematics, Kyushu University.

[Har94] Mary P. Harper. Storing logical form in a shared-packed forest. *Computational Linguistics*, 20(4):649–660, 1994.

[HR10] Muhammad Humayoun and Christophe Raffalli. Mathnat - mathematical text in a controlled natural language. In *Special issue: Natural Language Processing and its Applications*, volume 46 of *Journal on Research in Computing Science*. National Polytechnic Institute, Mexico, 2010.

[Hua08] Liang Huang. *Forest-based Algorithms in Natural Language Processing*. PhD thesis, University of Pennsylvania, 2008.

[HW06a] Helmut Horacek and Magdalena Wolska. Handling errors in mathematical formulas. In Mitsuru Ikeda, Kevin D. Ashley, and Tak-Wai Chan, editors, *Intelligent Tutoring Systems*, volume 4053 of *Lecture Notes in Computer Science*, pages 339–348. Springer, 2006.

[HW06b] Helmut Horacek and Magdalena Wolska. Interpreting semi-formal utterances in dialogs about mathematical proofs. *Data Knowl. Eng.*, 58(1):90–106, 2006.

[JFF02] Dean Jackson, Jon Ferraiolo, and Jun Fujisawa. Scalable vector graphics (svg) 1.1 specification. W3c candidate recommendation, World Wide Web Consortium (W3C), April 2002.

[Kam95] Hans Kamp. Discourse representation theory. In J. Verschueren, J.-O. Östman, and J. Blommaert, editors, *Handbook of Pragmatics*, pages 253–257. Benjamins, 1995.

[KK09] Andrea Kohlhase and Michael Kohlhase. Spreadsheet interaction with frames: Exploring a mathematical practice. In Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, editors, *MKM/Calculemus Proceedings*, number 5625 in LNAI, pages 341–356. Springer Verlag, July 2009.

[KMS99] Michael Kohlhase, Erica Melis, and Jörg Siekmann. ΩMEGA – a mathematical assistant. In Jelle Gerbrandy, Maarten Marx, Maarten de Rijke, and Yde Venema, editors, *Liber Amicorum for the Fiftieth Birthday of Johan van Benthem*, pages 248–251. ILLC, 1999.

[KMW04] Fairouz Kamareddine, Manuel Maarek, and Joe B. Wells. MathLang: An experience driven language of mathematics. In *Electronic Notes in Theoretical Computer Science 93C*, pages 138–160. Elsevier, 2004.

[KN11] Kevin Kofler and Arnold Neumaier. A dynamic generalized parser for common mathematical language. 2011.

[Koh04] Michael Kohlhase. Semantic markup for TEX/LATEX. In Paul Libbrecht, editor, *Mathematical User Interfaces*, 2004.

[Koh10] Michael Kohlhase. An open markup format for mathematical documents OMDoc [version 1.3]. Draft Specification, 2010.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[Lib10] Paul Libbrecht. Notations around the world: Census and exploitation. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *Intelligent Computer Mathematics*, number 6167 in LNAI, pages 398–410. Springer Verlag, 2010.

[Mat] Mathworld. `http://mathworld.wolfram.com`. seen March 2009.

[ME06] Wai Yin Mok and David W. Embley. Generating compact redundancyfree xml documents from conceptual-model hypergraphs. *IEEE Transactions on Knowledge and Data Engineering*, 18:2006, 2006.

[Mil] Bruce Miller. `LaTeXML`: A LaTeX to XML converter.

[NS10] Arnold Neumaier and Peter Schodl. A framework for representing and processing arbitrary mathematics. In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, pages 476–479, 2010.

[O'H05] K.L. O'Halloran. *Mathematical discourse: language, symbolism and visual images*. Continuum, 2005.

[Pak09] Scott Pakin. The Comprehensive LaTeX Symbol List. seen August 2011, 2009.

[PK11] Corneliu C. Prodescu and Michael Kohlhase. Mathwebsearch 0.5 - open formula search engine. sep 2011.

[Pla] PlanetMath.org – math for the people, by the people. `http://www.planetmath.org`. seen January 2011.

[Ran04] Aarne Ranta. Grammatical framework — a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189, 2004.

[RHJ98] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendation REC-html40, World Wide Web Consortium (W3C), April 1998.

[RK11] Florian Rabe and Michael Kohlhase. A scalable module system. Manuscript, submitted to Information & Computation, 2011.

[SGD+09] Heinrich Stamerjohanns, Deyan Ginev, Catalin David, Dimitar Misev, Vladimir Zamdzhiev, and Michael Kohlhase. Mathml-aware article conversion from LaTeX, a comparison study. In Petr Sojka, editor, *Towards Digital Mathematics Library, DML 2009 workshop*, pages 109–120. Masaryk University, Brno, 2009.

[SKG+10] Heinrich Stamerjohanns, Michael Kohlhase, Deyan Ginev, Catalin David, and Bruce Miller. Transforming large collections of scientific publications to XML. *Mathematics in Computer Science*, 3(3):299–307, 2010.

[Soc09] American Mathematical Society. Mathematics Subject Classification MSC2010. `http://www.ams.org/mathscinet/msc/`, 2009.

[Ste00] Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

[Sto] Michael Stoll. Lecture notes of courses taught at jacobs university.

[SW06] Elena Smirnova and Stephen M. Watt. Notation Selection in Mathematical Computing Environments. In *Proceddings Transgressive Computing 2006: A conference in honor of Jean Della Dora (TC 2006)*, pages 339–355, Granada, Spain, 2006.

[The02] The W3C HTML Working Group. Xhtml 1.0 the extensible hypertext markup language (second edition) – a reformulation of html 4 in xml 1.0. W3C recommendation, World Wide Web Consortium (W3C), 2002.

[Tra] Tralics: a LaTeX to XML translator.

[WAB06] Marc Wagner, Serge Autexier, and Christoph Benzmüller. PLATO: A Mediator between Text-Editors and Proof Assistance Systems. *7th Workshop on User Interfaces for Theorem Provers (UITP)*, 174(2):87–107, 2006.

[WG10] Magdalena Wolska and Mihai Grigore. Symbol declarations in mathematical writing: A corpus study. In Petr Sojka, editor, *Towards Digital Mathematics Library, DML workshop*, pages 119–127. Masaryk University, Brno, 2010.

[WGK11] Magdalena Wolska, Mihai Grigore, and Michael Kohlhase. Using discourse context to interpret object-denoting mathematical expressions. In Petr Sojka, editor, *Towards Digital Mathematics Library, DML workshop*. Masaryk University, Brno, 2011. in press.

[Wik] Table of Mathematical Symbols (from Wikipedia, the free encyclopedia). web page at `http://en.wikipedia.org/w/index.php?title=List_of_mathematical_symbols&oldid=446084263`. seen August 2011.

[Wik07] Wikipedia, the free encyclopedia. `http://www.wikipedia.org`, 2001–2007.

[WKK04] Magdalena Wolska and Ivana Kruijff-Korbayová. Analysis of mixed natural and symbolic input in mathematical dialogs. In *ACL*, pages 25–32, 2004.

[ZB11] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematics. submitted to International Journal of Document Analysis and Recognition, 2011.

[ZBM] Zentralblatt MATH. web page at `http://www.zentralblatt-math.org`. seen October 2009.

[Zin03] Claus Zinn. A computational framework for understanding mathematical discourse. *Logic Journal of the IGPL*, 11(4):457–484, 2003.

Appendix A

OpenCCG Grammar for Parsing Math Expressions

Below, we enclose the grammar which the author has developed using the concise syntax of the OpenCCG toolbox [CCG09]. A development version of the grammar will be maintained as part of the LaMaPUn library [Gin11], under the "Math-CCG" component of the Externals section.

A.1 Features

```
# A Grammar for parsing LaTeXML−based Math Expressions
# Designed as an external for the LaMaPUn OOPerl library
# Deyan Ginev, Magdalena Wolska

################## Features #################



feature {
  STRUCT<2>: expression { regular {invisible application} vector object[regular vector]}
             sequence{ list {nonempty} element } delimiter empty[list expression];
  FIXITY<2>: prefix infix postfix circumfix atomic;
  GLUE<2>: glue noglue;
  FENCE<2>: fenced {brace bracket bar parenthesis angular angularop
            frenchInterval russianInterval bra ket floor ceiling} unfenced;
  CASE<2>: upper lower;
  ontology: math−obj {
  ground {
        termobj {
              symbol
              scalar
              var
              operator {
```

```
                    prefix−op
                    infix−op
                    postfix−op
                }
                metaoperator {
                    prefix−meteaop
                    infix−metaop
                    postfix−metaop
                }
                relation {
                    prefix−rel
                    infix−rel
                    postfix−rel
                }
                metarelation {
                    prefix−metarel
                    infix−metarel
                    postfix−metarel
                }
                modifier [infix−rel infix−op]
            }
            formobj
        delim {
                fence
                punc {
                    separator
                }
        }
    }
    complex {
            termlist
            formlist
            exprlist
    }
    mid[termlist formlist exprlist punc]
    rich {
        interval
    }
    open[modifier fence interval] {
     openket
     openbra
     openfloor
     openceiling
    }
    close[modifier fence interval]
  };
}
```

A.2 Words

```
################## Words #################

#Declarations:

# Concatenation
word C:Concat;

# Numbers
word N1:Term(scalar):lower;
word N2:Term(scalar):lower;
...
word N19:Term(scalar):lower;
word N20:Term(scalar):lower;

# Upper-case terms
word T1:Term(math-obj):upper;
...
word T5:Term(math-obj):upper;

# Lower-case terms
word t1:Term(math-obj):lower;
...
word t5:Term(math-obj):lower;

# Upper-case Any
word A1:Any(math-obj);
...
word A10:Any(math-obj);

# Lower-case any
word a1:any(math-obj);
...
word a10:any(math-obj);

# Infix Operators: / *
word inop1:InfixOp;
...
word inop10:InfixOp;

# In the sense of division
word divide:InfixOp {
 ':';
};

# Ambigfix operators: + -
word ambigop1:AmbigfixOp;
...
word ambigop5:AmbigfixOp;
```

```
# Prefix operators
word preop1:PrefixOp; # Example − sin,cos
...
word preop5:PrefixOp;

# Postfix operators: ! %
word postop1:PostfixOp(postfix−op);
...
word postop10:PostfixOp(postfix−op);

# Higher prefix operators
word comp:Composition {
 ';';
 '∘';
 '⨾';
};

# Infix Relations: > < ≠ ≥ ≤ ≡
word inrel1:InfixRel;
...
word inrel10:InfixRel;
word '>':InfixRel;
word '<':InfixRel;

word equivalent:InfixRel(infix−rel) {
 '≡';
};

# Infix Metarelations: <=>, ∧, ∨
word inmetarel1:InfixMetaRel;
...
word inmetarel10:InfixMetaRel;

word '≡': InfixMetaRel;
word '=':Equality;
word formequality:InfixMetaRel {
 '=';
}

# Infix Dualtype:
word '→':ArrowDual;
word '⟶':ArrowDual;
word '↦':ArrowDual;
word '⇒':ArrowDual;
word '⇢':ArrowDual;
word '↪':ArrowDual;

word barop:AmbigfixOp {
 '|'; # For manifold constructors, restrictions, ranges etc.
};
```

```
# "Such that" operators
word '|':Modifier;
word '|':Modifier;
word ':':Modifier;
word ';':Modifier;
word assignment:Modifier {
 '=';
};

############ FENCES ############

word '(':Paranthesis(open);
word '[':Bracket(open);
word '{':Brace(open);
word ']':FrenchInterval(open);
word langle:AngularBracket(open) {
 '⟨';
};
word opangle:OpAngularBracket(open) {
 '<';
};

word bar:Bar(open) {
 '|';
}

word russianInterval:RussianInterval(open) {
 '(';
 '[';
}

word bra:Bra(open) {
 '⟨';
};
word ket:Ket(open) {
 '|';
};

word '⌊':Floor;
word '⌈':Ceiling;

word close:Close {
 ')' :parenthesis;
 ']' :bracket;
 '}' :brace;
 '[' :frenchInterval;
 ')' :russianInterval;
 ']' :russianInterval;
 '|' :bar;
 '⟩' :angular;
 '>' :angularop;
```

```
 '⌋' :floor;
 '⌉' :ceiling;
}

word closebra:Close {
 '|':bra;
}
word closeket:Close {
 '⟩' :ket;
}

word separator:ListC(mid) {
 ',';
 ';';
};
word '.':Punc;
```

A.3 Categories

```
################## Categories ################

family Term {
 entry: term<2>[unfenced atomic object X]:
        X:termobj(*);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

family Form {
 entry: form<2>[unfenced atomic object noglue X]:
        X:formobj(*);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

family InfixOp {
 # Expressions:
 entry: term<2>[unfenced inop application noglue X0]/*term[fenced expression noglue X1]
                                                        \*term[expression noglue X2]:
                X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[unfenced inop application noglue X0]/*term[unfenced prefix expression noglue X1]
                                                        \*term[expression noglue X2]:
                X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[unfenced inop application noglue X0]/*term[unfenced postfix expression noglue X1]
                                                        \*term[expression noglue X2]:
                X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
```

```
 entry: term<2>[unfenced inop application noglue X0]/*term[unfenced atomic expression noglue X1]
                                                      \*term[expression noglue X2]:
        X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[unfenced inop application noglue X0]/*term[unfenced invisible noglue X1]
                                                      \*term[expression noglue X2]:
        X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);

 # Lists:
 entry: term<2>[unfenced inop nonempty noglue X0]/*term[fenced list noglue X1]
                                                    \*term[fenced list noglue X2]:
        X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

family PrefixOp {
 # We need associative modality to support chaining of prefix operators (e.g. for higher-order equality)
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression infix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression prefix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression postfix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression circumfix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression atomic noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression fenced noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);

 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

family PostfixOp {
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression infix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression prefix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression postfix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression circumfix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression atomic noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression fenced noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
```

```
        X0:termobj(*);
}

family AmbigfixOp { # infix, postfix and prefix
 entry: term<2>[unfenced infix application X0]/*term[expression noglue X1]\*term[expression noglue X2]:
                X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[unfenced infix nonempty noglue X0]/*term[fenced list noglue X1]
                                                    \*term[fenced list noglue X2]:
                X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);

 # Prefix
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression infix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression prefix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression postfix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression circumfix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression atomic noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/^term[expression fenced noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);

 # Postfix
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression infix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression prefix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression postfix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression circumfix noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression atomic noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
 entry: term<2>[postfix unfenced application noglue X0]\*term[expression fenced noglue X1]:
        X0:postfix-op(* <Left-Arg>X1);
}

family Composition {
# Standard, function composition in applied context
 entry: (term[X0]/*term[expression])/*(term[prefix X1]/*term[expression])
                                     \*(term[prefix X2]/*term[expression]):
        X0:infix-metaop(* <Left-Arg>X2 <Right-Arg>X1);

# Function composition in "object" context, e.g. S ∘ R = foo
 entry: term[noglue element infix unfenced X0]/*(term[prefix X1]/*term[expression])
```

```
                                                \*(term[prefix X2]/*term[expression]):
        X0:infix-metaop(* <Left-Arg>X2 <Right-Arg>X1);
# Can be used as term in sequence context
 entry: term[noglue element infix unfenced X0]/*term[element X1]\*term[element X2]:
        X0:infix-metaop(* <Left-Arg>X2 <Right-Arg>X1);
}

family Concat {
 entry: term<2>[infix unfenced invisible X0]/^term[noglue atomic object X1]\*term[noglue object X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue atomic object X1]\*term[noglue fenced X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue atomic object X1]
                                                \*term[noglue unfenced prefix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue atomic object X1]
                                                \*term[noglue unfenced postfix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue atomic object X1]
                                                \*term[noglue unfenced circumfix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);

 entry: term<2>[infix unfenced invisible X0]/^term[noglue invisible X1]\*term[noglue atomic object X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue invisible X1]\*term[noglue fenced X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue invisible X1]
                                                \*term[noglue unfenced prefix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue invisible X1]
                                                \*term[noglue unfenced postfix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);

 entry: term<2>[infix unfenced invisible X0]/^term[noglue invisible X1]
                                                \*term[noglue unfenced circumfix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);


 entry: term<2>[infix unfenced invisible X0]/^term[noglue fenced X1]\*term[noglue atomic object X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue fenced X1]\*term[noglue fenced X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue fenced X1]
                                                \*term[noglue unfenced prefix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue fenced X1]
                                                \*term[noglue unfenced postfix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue fenced X1]
                                                \*term[noglue unfenced circumfix application X2]:
        X0:infix-op(invisible-op <Left-Arg>X2 <Right-Arg>X1);
```

```
 entry: term<2>[infix unfenced invisible X0]/^term[noglue unfenced prefix application X1]
                                                \*term[noglue atomic object X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue unfenced prefix application X1]
                                                \*term[noglue fenced X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue unfenced postfix application X1]
                                                \*term[noglue atomic object X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue unfenced postfix application X1]
                                                \*term[noglue fenced X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue unfenced circumfix application X1]
                                                \*term[noglue atomic object X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[infix unfenced invisible X0]/^term[noglue unfenced circumfix application X1]
                                                \*term[noglue fenced X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);

 # Modifiers such as l(l\geq q) or l(\geq q)
 entry: term<~2>[infix unfenced invisible X0]/*form[noglue regular fenced X1]
                                                \*term<2>[noglue atomic object X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);
 entry: term<~2>[infix unfenced invisible X0]/*(form[noglue regular fenced X1]
                                                \*term[expression noglue])\*term<2>[noglue atomic object X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);

 #Support infix Any's with postfix and prefix concats:
 entry: term<~2>[glue X0]/*term<2>[fenced noglue X1]:
        X0:prefix−op(invisible−op <Right−Arg>X1);
 entry: term<~2>[glue X0]/*term<2>[atomic object noglue X1]:
        X0:prefix−op(invisible−op <Right−Arg>X1);
 entry: term<~2>[glue X0]\*term<2>[fenced noglue X1]:
        X0:postfix−op(invisible−op <Left−Arg>X1);
 entry: term<~2>[glue X0]\*term<2>[atomic object noglue X1]:
        X0:postfix−op(invisible−op <Left−Arg>X1);

 # Preserve elements
 entry: term<2>[atomic unfenced element noglue X0]/^term[noglue atomic element unfenced X1]
                                                \*term[noglue atomic element unfenced X2]:
        X0:infix−op(invisible−op <Left−Arg>X2 <Right−Arg>X1);

}

###### Relations ######

family InfixRel {
 entry: form<2>[infix infix−rel unfenced application noglue X0]/*term[expression noglue X1]
                                                \*term[expression noglue X2]:
```

```
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);
 entry: form<2>[infix infix−rel unfenced application noglue X0]\∗term[expression noglue X2]
                                                                    /∗term[expression noglue X1]:
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);

# Chaining of infix relations 2 > 3 > 4
 entry: form<2>[infix infix−rel unfenced application noglue X0]/∗term[expression noglue X1]
                                                                    \∗form[expression infix infix−rel noglue X2]:
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);

 # Lists:
 entry: form<2>[infix infix−rel unfenced nonempty noglue X0]/∗term[fenced list noglue X1]
                                                                    \∗term[fenced list noglue X2]:
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);
 entry: form<2>[infix infix−rel unfenced nonempty noglue X0]/∗term[fenced list noglue X1]
                                                                    \∗form[fenced list infix infix−rel noglue X2]:
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);
# List of operands e.g. a,b,c \in S
 entry: form<2>[infix infix−rel unfenced noglue X0]/∗term[expression noglue X1]
                                                        \∗term[unfenced list noglue X2]:
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);

# Flexibly become modifiers (for objects, lists and fenced expressions):
# "which is" statements
 entry: term<2>[unfenced object noglue X0]/∗term[expression noglue X1]\∗term[unfenced object noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[fenced expression noglue X0]/∗term[expression noglue X1]\∗term[fenced expression noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
 entry: term<2>[list noglue X0]/∗term[expression noglue X1]\∗term[list noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);

#Second−order relations:
entry: form<2>[infix infix−rel unfenced application noglue X0]/∗(term[X1]/term)\∗(term[X2]/term):
       X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);
entry: form<2>[infix infix−rel unfenced application noglue X0]/∗(term[X1]/term)
                                                                    \∗form[expression infix infix−rel noglue X2]:
       X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);


 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(∗);
}

family PrefixRel{
 entry: form<2>[prefix unfenced application noglue X0]/^term[expression noglue X1]:
        X0:prefix−rel(∗ <Right−Arg>X1);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(∗);
}
```

```
family PostfixRel {
 entry: form<2>[postfix unfenced application noglue X0]\*term[expression noglue X1]:
        X0:postfix-rel(* <Left-Arg>X1);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

###### Meta Relations ######

family InfixMetaRel {
entry: form<2>[infix unfenced application infix-metarel noglue X0]/*form[fenced expression noglue X1]
                                                                    \*form[expression noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
entry: form<2>[infix unfenced application noglue X0]/*form[unfenced prefix expression noglue X1]
                                                      \*form[expression noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
entry: form<2>[infix unfenced application infix-metarel noglue X0]
                                                /*form[unfenced infix infix-rel expression noglue X1]
                                                \*form[expression noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
entry: form<2>[infix unfenced application infix-metarel noglue X0]
                                                /*form[unfenced postfix expression noglue X1]
                                                \*form[expression noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
entry: form<2>[infix unfenced application infix-metarel noglue X0]
                                                /*form[unfenced atomic expression noglue X1]
                                                \*form[expression noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
entry: form<2>[infix unfenced application infix-metarel noglue X0]
                                                /*form[unfenced invisible noglue X1]
                                                \*form[expression noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);

# Lists:
entry: form<2>[infix unfenced nonempty noglue X0]/*form[fenced list noglue X1]
                                                   \*form[fenced list noglue X2]:
        X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);

 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

family PrefixMetaRel {
entry: form<2>[prefix unfenced application noglue X0]/*form[expression noglue X1]:
        X0:prefix-metarel(* <Right-Arg>X1);
 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
```

```
337         X0:termobj(*);
    }

    family PostfixMetaRel {
    entry: form<2>[infix unfenced application noglue X0]\*form[expression noglue X2]:
342         X0:infix-metarel(* <Left-Arg>X2);
     # Can be used as objects in sequence context
     entry: term<2>[unfenced atomic element noglue X0]:
            X0:termobj(*);
    }
347
    family AmbigfixMetaRel {
    entry: form<2>[infix unfenced application noglue X0]/*form[expression noglue X1]
                                                          \*form[expression noglue X2]:
            X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
352 entry: form<2>[prefix unfenced application noglue X0]/*form[expression noglue X1]:
            X0:prefix-metarel(* <Right-Arg>X1);
    entry: form<2>[infix unfenced application noglue X0]\*form[expression noglue X2]:
            X0:infix-metarel(* <Left-Arg>X2);

357  # Can be used as objects in sequence context
     entry: term<2>[unfenced atomic element noglue X0]:
            X0:termobj(*);
    }

362
    ###### Duals ######

    family ArrowDual {
    entry: term<2>[inop unfenced element noglue X0]/*term[expression noglue X1]\*term[expression noglue X2]:
367         X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
     entry: form<2>[infix unfenced element noglue X0]/*form[expression noglue X1]\*form[expression noglue X2]:
            X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);
    entry: term<2>[inop unfenced element noglue X0]/*term[expression noglue X1]\*term[element noglue X2]:
            X0:infix-op(* <Left-Arg>X2 <Right-Arg>X1);
372  entry: form<2>[infix unfenced element noglue X0]/*form[expression noglue X1]\*form[element noglue X2]:
            X0:infix-metarel(* <Left-Arg>X2 <Right-Arg>X1);

     # Can be used as objects in sequence context
     entry: term<2>[unfenced atomic element noglue X0]:
377         X0:termobj(*);
    }

    family Equality {
      #Term equality
382   entry: form<2>[infix infix-rel unfenced application noglue X0]/*term[noglue X1]\*term[noglue X2]:
            X0:infix-rel(* <Left-Arg>X2 <Right-Arg>X1);
      entry: form<2>[infix infix-rel unfenced application noglue X0]/*term[noglue X1]
                                                          \*form[expression infix infix-rel noglue X2]:
            X0:infix-rel(* <Left-Arg>X2 <Right-Arg>X1);
387
```

```
  #Second−order equality:
  entry: form<2>[infix infix−rel unfenced application noglue X0]/∗(term[X1]/term)\∗(term[X2]/term):
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: form<2>[infix infix−rel unfenced application noglue X0]/∗(term[X1]/term)
                                                          \∗form[expression infix infix−rel noglue X2]:
        X0:infix−rel(∗ <Left−Arg>X2 <Right−Arg>X1);

  #Formula equality
  entry: form<2>[infix infix−metarel unfenced application noglue X0]/∗form[noglue X1]\∗form[noglue X2]:
        X0:infix−metarel(∗ <Left−Arg>X2 <Right−Arg>X1);

  # Can be used as objects in sequence context
  entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(∗);
}

family Modifier {
  entry: term<2>[infix unfenced element noglue X0]/∗term[sequence noglue X1]
                                                    \∗term[unfenced element noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: term<2>[infix unfenced object noglue X0]/∗term[sequence noglue X1]
                                                   \∗term[unfenced object noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: term<2>[infix fenced sequence noglue X0]/∗term[sequence noglue X1]
                                                  \∗term[fenced sequence noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: term<2>[infix fenced object noglue X0]/∗term[sequence noglue X1]
                                                 \∗term[fenced object noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);

  entry: term<2>[infix unfenced element noglue X0]/∗form[noglue X1]\∗term[unfenced element noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: term<2>[infix unfenced object noglue X0]/∗form[noglue X1]\∗term[unfenced object noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: term<2>[infix fenced sequence noglue X0]/∗form[noglue X1]\∗term[fenced sequence noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);
  entry: term<2>[infix fenced object noglue X0]/∗form[noglue X1]\∗term[fenced object noglue X2]:
        X0:modifier(∗ <Left−Arg>X2 <Right−Arg>X1);

  # Can be used as objects in sequence context
  entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(∗);
}

########### Fences ############

family Paranthesis {
 entry: term<~2>[parenthesis vector noglue X0]/∗close[parenthesis X1]/∗term<2>[noglue sequence X2]:
        X0:open(∗ <Arg>X2 <Close>X1);
 entry: form<~2>[parenthesis vector noglue X0]/∗close[parenthesis X1]/∗form<2>[noglue sequence X2]:
```

```
        X0:open(* <Arg>X2 <Close>X1);
 entry: term<~2>[parenthesis regular noglue X0]/*close[parenthesis X1]/*term<2>[noglue expression X2]:
        X0:open(* <Arg>X2 <Close>X1);
 entry: form<~2>[parenthesis regular noglue X0]/*close[parenthesis X1]/*form<2>[noglue expression X2]:
        X0:open(* <Arg>X2 <Close>X1);

 # Empty parenthesis:
 entry: term[empty parenthesis noglue X0]/*close[parenthesis X1]:
        X0:open(* <Arg>empty <Close>X1);

 entry: (form<~2>[parenthesis regular noglue X0]\*term)/*close[parenthesis X1]
                                                        /*(form<2>[noglue expression X2]\*term):
        X0:open(* <Arg>X2 <Close>X1);
}

family Bracket {
 entry: term<~2>[bracket expression noglue X0]/*close[bracket X1]/*term<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 entry: form<~2>[bracket expression noglue X0]/*close[bracket X1]/*form<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 # Empty brackets:
 entry: term[empty bracket noglue X0]/*close[bracket X1]:
        X0:open(* <Arg>empty <Close>X1);
}

family Brace {
 entry: term<~2>[brace expression noglue X0]/*close[brace X1]/*term<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 entry: form<~2>[brace expression noglue X0]/*close[brace X1]/*form<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 # Empty braces:
 entry: term[empty brace noglue X0]/*close[brace X1]:
        X0:open(* <Arg>empty <Close>X1);
}

family AngularBracket {
 entry: term<~2>[angular expression noglue X0]/*close[angular X1]/*term<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 entry: form<~2>[angular expression noglue X0]/*close[angular X1]/*form<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 # Empty angular brackets:
 entry: term[empty angular noglue X0]/*close[angular X1]:
        X0:open(* <Arg>empty <Close>X1);
}

family OpAngularBracket {
 entry: term<~2>[opangular expression noglue X0]/*close[opangular X1]/*term<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 entry: form<~2>[opangular expression noglue X0]/*close[opangular X1]/*form<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);
 # Empty angular brackets:
```

```
 entry: term[empty opangular noglue X0]/*close[opangular X1]:
        X0:open(* <Arg>empty <Close>X1);
}


family Bra {
 entry: term<~2>[bra object noglue X0]/*close[bra X1]/*term<2>[noglue X2]:
        X0:openbra(* <Arg>X2 <Close>X1);
}

family Ket {
 entry: term<~2>[ket object noglue X0]/*close[ket X1]/*term<2>[noglue X2]:
        X0:openket(* <Arg>X2 <Close>X1);
}

family FrenchInterval {
 entry: term<~2>[frenchInterval object noglue X0]/*close[frenchInterval X1]/*term<2>[nonempty noglue X2]:
        X0:interval(* <Arg>X2 <Close>X1);
}

family RussianInterval {
 entry: term<~2>[russianInterval object noglue X0]/*close[russianInterval X1]/*term<2>[nonempty noglue X2]:
        X0:interval(* <Arg>X2 <Close>X1);
}

family Floor {
 entry: term<~2>[floor object noglue X0]/*close[floor X1]/*term<2>[expression noglue X2]:
        X0:openfloor(* <Arg>X2 <Close>X1);
}

family Ceiling {
 entry: term<~2>[ceiling object noglue X0]/*close[ceiling X1]/*term<2>[expression noglue X2]:
        X0:openceiling(* <Arg>X2 <Close>X1);
}

family Bar {
 entry: term<~2>[bar expression noglue X0]/*close[bar X1]/*term<2>[noglue X2]:
        X0:open(* <Arg>X2 <Close>X1);

 # Empty bars:
 entry: term[empty bar noglue X0]/*close[bar X1]:
        X0:open(* <Arg>empty <Close>X1);
}


family Close {
 entry: close<2>[ noglue X]:
   X:close(*);
}

######### Lists ########
```

```
family ListC {
 entry: term[unfenced infix nonempty X]/*term[X2]\*term[X1]:
        X:termlist(* <Left-Arg>X1 <Right-Arg> X2);
 entry: form[unfenced infix nonempty X]/*form[X2]\*form[X1]:
        X:formlist(* <Left-Arg>X1 <Right-Arg> X2);
# We know this is a meta separation if the types don't match:
# Even add a terminal type "expr" !
 entry: expr[unfenced infix nonempty X]/*form[X2]\*term[X1]:
        X:exprlist(* <Left-Arg>X1 <Right-Arg> X2);
 entry: expr[unfenced infix nonempty X]/*term[X2]\*form[X1]:
        X:exprlist(* <Left-Arg>X1 <Right-Arg> X2);
 entry: expr[unfenced infix nonempty X]/*form[X2]\*expr[X1]:
        X:exprlist(* <Left-Arg>X1 <Right-Arg> X2);
 entry: expr[unfenced infix nonempty X]/*term[X2]\*expr[X1]:
        X:exprlist(* <Left-Arg>X1 <Right-Arg> X2);
}


family Punc {
  entry: punc<2>[unfenced delimiter noglue X]:
        X:punc(*);

 # Postfix operator that wraps-up an expression:
 entry: expr[unfenced postfix nonempty X0]\*term[X1]: X0:punc(X1);
 entry: expr[unfenced postfix nonempty X0]\*form[X1]: X0:punc(X1);

 # Can be used as objects in sequence context
 entry: term<2>[unfenced atomic element noglue X0]:
        X0:termobj(*);
}

######### Any #######

family Any {
 entry: term[upper unfenced atomic object noglue X]:
        X:termobj(*);
 entry: form[upper unfenced atomic object noglue X]:
        X:formobj(*);
 entry: term[upper unfenced atomic element noglue X]:
        X:termobj(*);
 entry: form[upper unfenced atomic element noglue X]:
        X:formobj(*);

 entry: punc[upper unfenced atomic object delim noglue X]:
        X:punc(*);

 entry: term<2>[prefix unfenced application noglue X0]/*term[expression infix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
 entry: term<2>[prefix unfenced application noglue X0]/*term[expression prefix noglue X1]:
        X0:prefix-op(* <Right-Arg>X1);
```

```
592  entry: term<2>[prefix unfenced application noglue X0]/*term[expression circumfix noglue X1]:
           X0:prefix-op(* <Right-Arg>X1);
     entry: term<2>[prefix unfenced application noglue X0]/*term[expression atomic noglue X1]:
           X0:prefix-op(* <Right-Arg>X1);
     entry: term<2>[prefix unfenced application noglue X0]/*term[expression fenced noglue X1]:
597        X0:prefix-op(* <Right-Arg>X1);

     entry: term<2>[infix unfenced application infix-op noglue X0]/*term[lower expression X1]
                                                                  \*term[lower expression X2]:
           X0:infix-op ( * <Left-Arg> X2 <Right-Arg> X1 );
602
     entry: form<2>[prefix unfenced application noglue X0]/*term[noglue expression X1]:
           X0:prefix-rel ( * <Right-Arg> X1 );
     entry: form<2>[infix unfenced application noglue X0]/*term[lower expression X1]\*term[lower expression X2]:
           X0:infix-rel ( * <Left-Arg> X2 <Right-Arg> X1 );
607  entry: form<2>[infix unfenced application noglue X0]/*term[lower expression X1]
                                                        \*form[infix infix-rel lower expression X2]:
           X0:infix-rel ( * <Left-Arg> X2 <Right-Arg> X1 );
     }

612  family any {
     entry: term[lower unfenced atomic object noglue X]:
           X:termobj(*);
     entry: form[lower unfenced atomic object noglue X]:
           X:formobj(*);
617  entry: term[lower unfenced atomic element noglue X]:
           X:termobj(*);
     entry: form[lower unfenced atomic element noglue X]:
           X:formobj(*);

622  entry: punc[lower unfenced atomic object noglue X]:
           X:punc(*);

     entry: term<2>[prefix unfenced application noglue X0]/*term[expression infix noglue X1]:
           X0:prefix-op(* <Right-Arg>X1);
627  entry: term<2>[prefix unfenced application noglue X0]/*term[expression prefix noglue X1]:
           X0:prefix-op(* <Right-Arg>X1);
     entry: term<2>[prefix unfenced application noglue X0]/*term[expression circumfix noglue X1]:
           X0:prefix-op(* <Right-Arg>X1);
     entry: term<2>[prefix unfenced application noglue X0]/*term[expression atomic noglue X1]:
632        X0:prefix-op(* <Right-Arg>X1);
     entry: term<2>[prefix unfenced application noglue X0]/*term[expression fenced noglue X1]:
           X0:prefix-op(* <Right-Arg>X1);

     entry: term<2>[infix unfenced application infix-op noglue X0]/*term[upper expression X1]
637                                                               \*term[upper expression X2]:
           X0:infix-op ( * <Left-Arg> X2 <Right-Arg> X1 );

     entry: form<2>[prefix unfenced application prefix-rel noglue X0]/*term[ noglue expression X1]:
           X0:prefix-rel ( * <Right-Arg> X1 );
642  entry: form<2>[infix unfenced application noglue X0]/*term[upper expression X1]\*term[upper expression X2]:
```

```
        X0:infix−rel ( ∗ <Left−Arg> X2 <Right−Arg> X1 );
 entry: form<2>[infix unfenced application noglue X0]/∗term[upper expression X1]
                                                        \∗form[upper infix infix−rel expression X2]:
        X0:infix−rel ( ∗ <Left−Arg> X2 <Right−Arg> X1 );
}
```

A.4 Test Sentences

```
################## Test sentences #################
testbed {
# Analytic:
  N1: 2; # A number in an application and list contexts (2 parses)
  aa: 0;
  a1 inop1 a2 inop2 a3: 1;
  a1 inop1 ambigop1 a2: 1;
  a1 C a2 inop1 a3 : 1;
  a1 C A1 C a1 : 3; # Reflexivity with relation A1, or just a concat tree, or a 3−letter object name
  A1 C a1 C a2 C a3 C a4 :1; # Prefix application of A1
  a1 C A1 C A2 C A3 C A4 :1; # Prefix application of a1
  ']' a1 ',' a2 '[' : 1; # French interval
  '[' a1 ',' a2 ']' : 3; # Russian interval or ambig brackets (term,form)
  '(' a1 ',' a2 ']' : 1; # Russian interval
  '{' a1 ',' a2 '}' : 2; # Ambig braces (term, form)
  '{' '}' : 1; # Empty Set
  a1 C '(' ')' : 1; # Function with no arguments (term)
  a1 C '(' ']' : 0; # Ungrammatical nonsence
  '(' A1 ',' inop1 ')' : 2; # Group constructor notation (2 parses, since RussianInterval kicks in)
  N1 C a1 postop1 : 2; # Factorial of concat(N1,a1) or concat(N1,factorial(a1)) ?
  a1 C '(' a2 ',' a3 '=' t1 inop1 t2 ')' '=' a3 : 2; # Dual use of equality (2 parses − interval or args)
'{' a1 ':' a1 inrel1 A1 '}' : 4;# Set constructor, a1 modified by rest, or a1:a1 related to A1, division?
A1 inmetarel1 A2 '=' A3 inmetarel2 A4 :2; # Equality as a metarel and as a rel

# Experimental:

# Document 1

preop1 A1 inrel1 t1 C preop1 A1 C A1 : 2; # Train 1
a1 '=' '(' A4 ',' N1 ')' : 1;
'[' A2 C a1 C a1 C '(' A1 ')' inop1 A1 ']' :1;
# Triple attachment ambiguity due to bar and equivalence:
'{' '(' A2 ',' a2 ')' '|' a2 '≡' a1 ambigop1 preop1 '(' A2 ambigop1 A2 ')' '}' :3;

#Source: (X,\tilde{c}W)\in\tilde{\mathcal{F}I}(\mathcal{V}):
'(' A1 ',' a1 C A1 ')' inrel1 a1 C '(' A1 ')' :1;
'⟨' '⟨' inop1 ',' inop1 '⟩' '⟩' ':' a1 inop1 a1 '→' A1 : 1;
a1 C '(' a1 inrel1 a1 ')' :1;
preop1 '∘' a1 :2; #Function composition (applicable or made atomic)
A1 '∘' A2 '=' '{' '(' a1 ',' a2 ')' '|' a1 inrel1 A1 '}' :1; #Function composition
```

```
A1 inrel1 A1 C A4 ',' a1 '=' N1 ',' N1 :2; # Two ways to split the comas
a1 inop1 a1 '=' '(' a1 ';' ambigop1 a1 ')' :1;
inop1 a2 :0; # operation without left argument
inrel1 a2 :1; # modifier without left argument
A1 '∘' A1 C '(' a1 ')' '∘' A1 '=' A1 C '(' a2 ')' :1;

# Document 2

a1 '=' '|' a1 '|' : 2; # Assignment or equality (always terms)
a1 C '(' a1 ')' '=' '|' N1 '⟩' C '⟨' N1 '|' :1; #Bra−ket notation
 # Higher−equality of partial differential funct:
preop1 '=' a1 C preop1 preop1 :1;
   # \partial _{x}^{2}=\eta^{{\mu\nu}}\partial _{\mu}\partial _{\nu}
ambigop1 ambigop1: 1; #Chained parse
A2 C '(' inrel1 N1 ')' :1; #M^{{b}}_{{21}}(\neq 0)
a1 '=' '(' N1 inop1 N1 ',' N1 inop1 N1 inop1 ',' N1 inop1 N1 ',' N1 inop1 N1 ')' :0;
#\boldsymbol{\theta}=(1/3,1/7/,1/7,6/7) *ungrammatical trailing slash
'(' A1 ',' a2 ')' '=' '(' A1 ',' '|' '|' ')' :1;

# Document 6
a1 inrel1 A1 ',' a2 inrel1 A1 inop1 '{' N1 '}' :2; #Ambiguous modifier scope| a \in R, b \in R \ {0}
a1 ',' a2 ',' a3 inrel1 A1 :2; # a,b,c \in S #modifies last element or entire list
a1 '=' preop1 a1 '|' :1;
preop1 '≡' a2 C preop1 :1;


# Document 10
'(' '<' a1 '>' ',' A2 ',' a1 ')' :1; #smaller and greater used as fences
a1 C a1 '∘' a1 ':' a1 '→' A1 :1;# Two−symbol object needs to be produced by concat
'⌊' a1 inop1 N1 '⌋':1; # Floor
}
```

Appendix B

Grammar Evaluation Results

B.1 Grammatical Parses

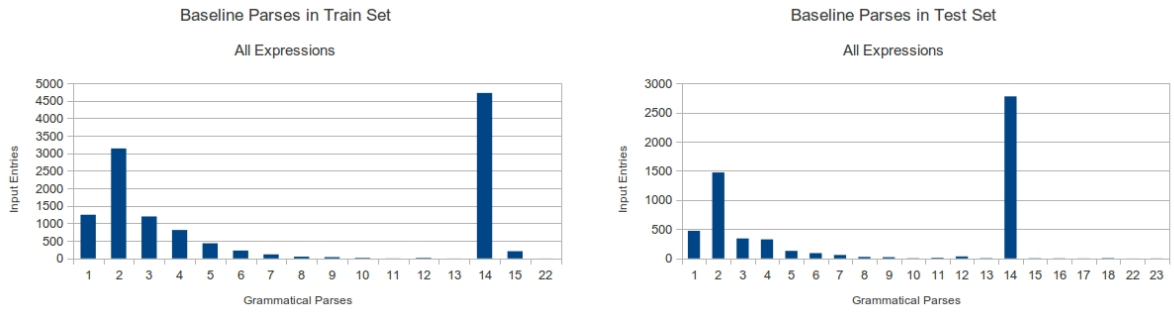


Figure B.1: Number of Grammatical Parses over All Baseline Expressions

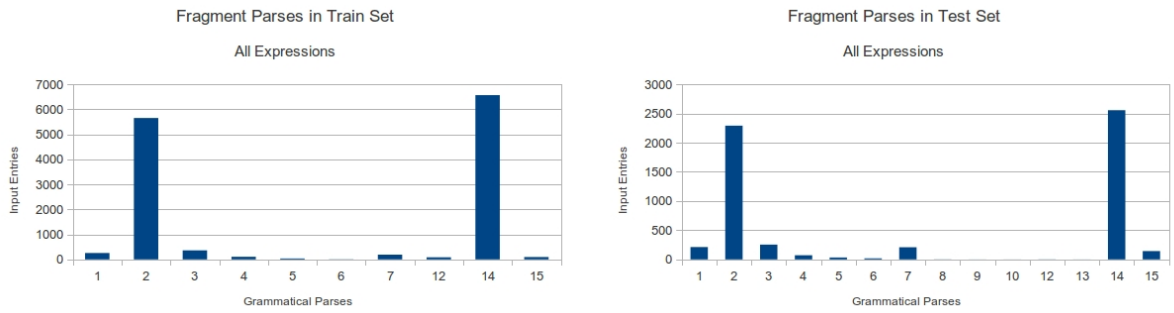


Figure B.2: Number of Grammatical Parses over All Fragment Expressions

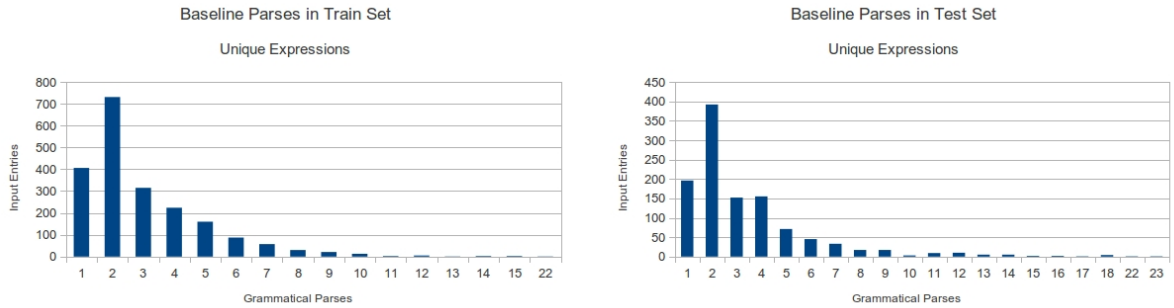


Figure B.3: Number of Grammatical Parses over Unique Baseline Expressions

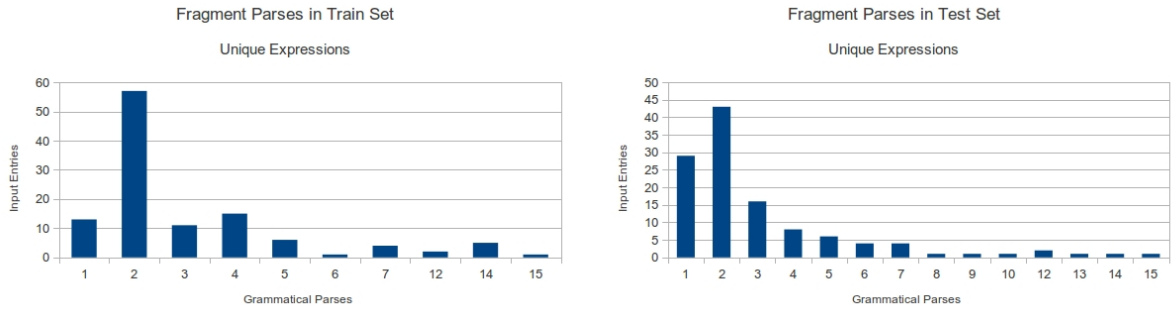


Figure B.4: Number of Grammatical Parses over Unique Fragment Expressions

B.2 Expression Length

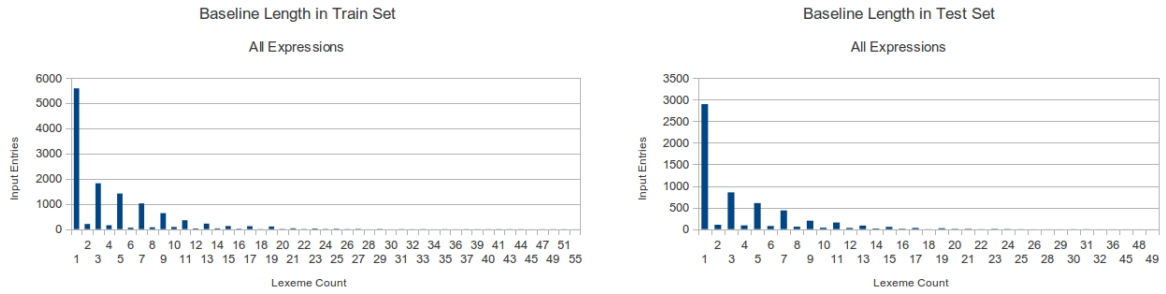


Figure B.5: Lexeme Count over All Baseline Expressions

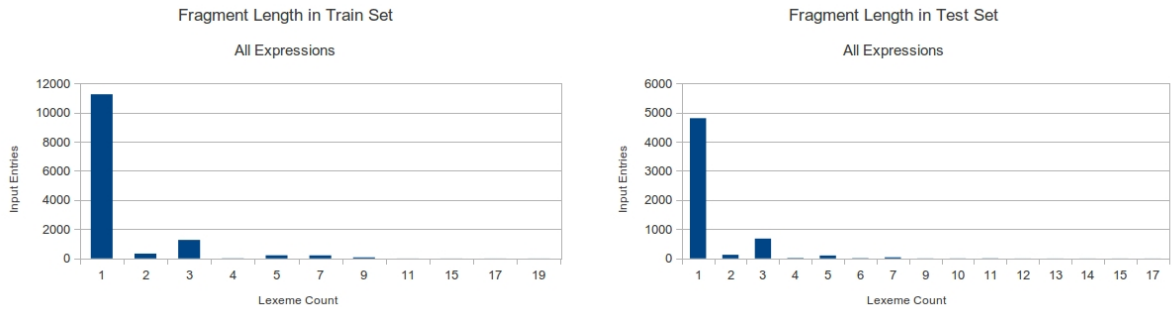


Figure B.6: Lexeme Count over All Fragment Expressions

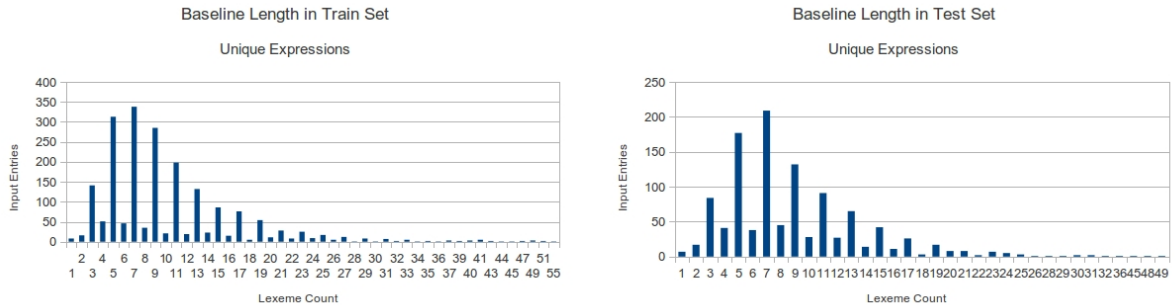


Figure B.7: Lexeme Count over Unique Baseline Expressions

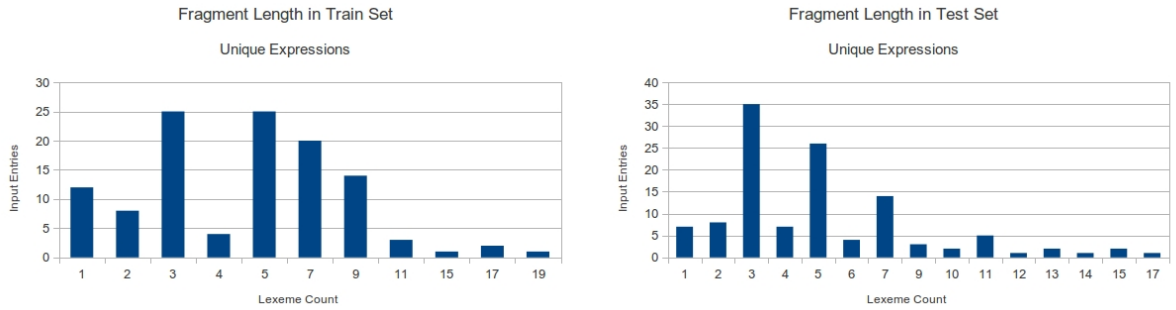


Figure B.8: Lexeme Count over Unique Fragment Expressions

B.3 Average Parses per Expression Length

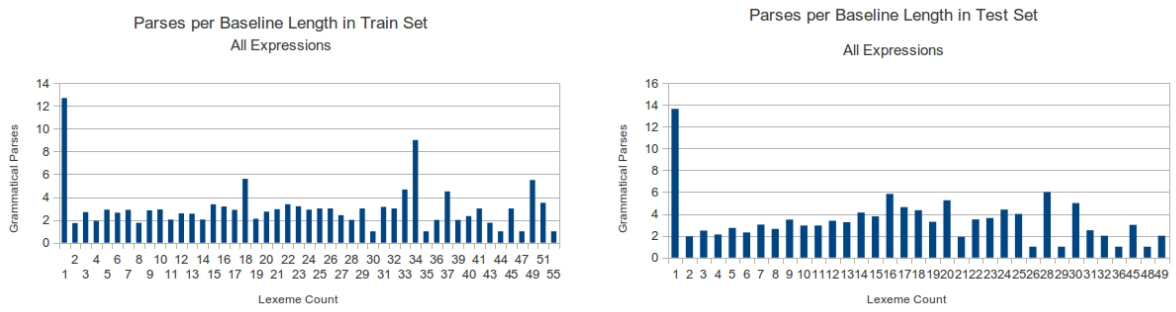


Figure B.9: Parses per Lexeme Count over All Baseline Expressions

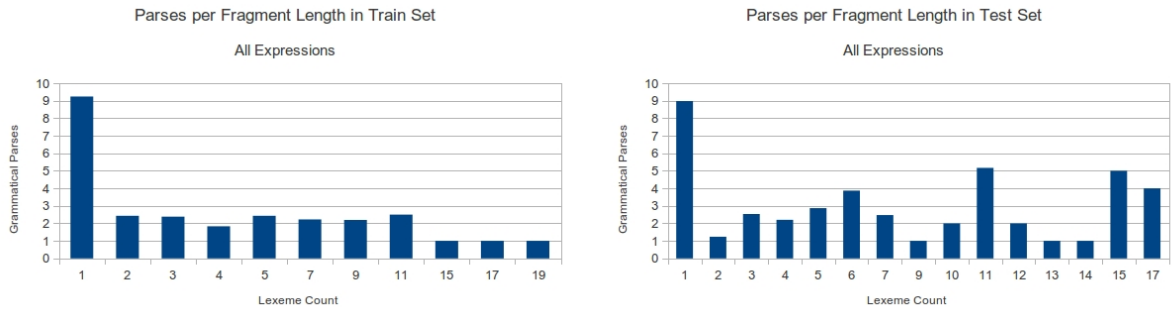


Figure B.10: Parses per Lexeme Count over All Fragment Expressions

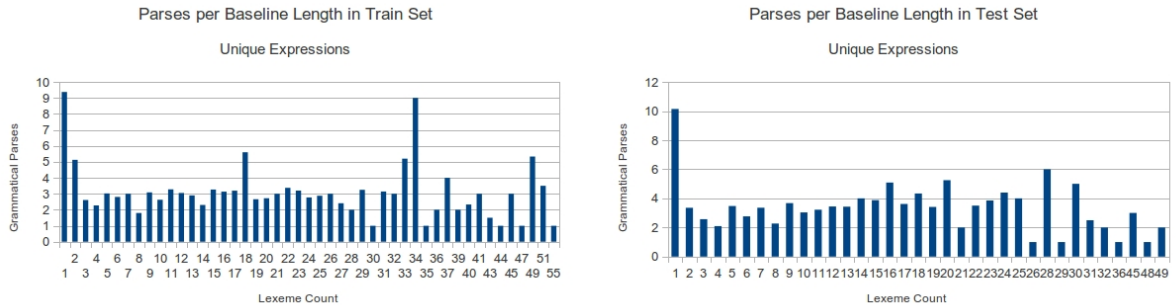


Figure B.11: Parses per Lexeme Count over Unique Baseline Expressions

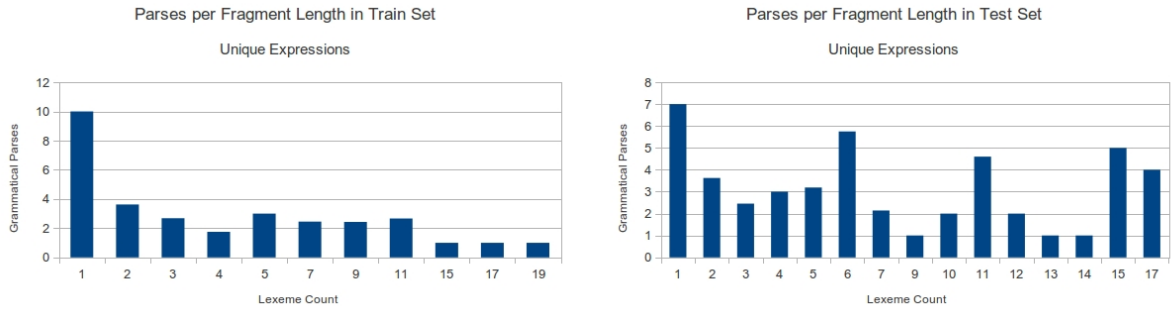


Figure B.12: Parses per Lexeme Count over Unique Fragment Expressions

Appendix C

A Running Example of Pipeline Processing

C.1 LaTeX Input

```
$ x , y \in S $
```

C.2 LaTeXML Noparse XML

```
<Math mode="inline" tex="x,y\in S" xml:id="p1.m1">
  <XMath>
    <XMTok role="UNKNOWN" font="italic">x</XMTok>
    <XMTok role="PUNCT">,</XMTok>
    <XMTok role="UNKNOWN" font="italic">y</XMTok>
    <XMTok meaning="element-of" name="in" role="RELOP">∈</XMTok>
    <XMTok role="UNKNOWN" font="italic">S</XMTok>
  </XMath>
</Math>
```

C.3 OpenCCG Input

```
a1 , a2 inrel1 A1
```

C.4 Grammatical Parses

```
5 parses found.

Parse 1: term<27>{GLUE=noglue, STRUCT=list, index=X0_42:modifier} :
  @i1:modifier(inrel1 ^
               <Left-Arg>(s1:mid ^ separator ^
                          <Left-Arg>(a1:termobj ^ a1) ^
                          <Right-Arg>(a2:termobj ^ a2)) ^
               <Right-Arg>(a3:termobj ^ a3))
Parse 2: form{FENCE=unfenced, FIXITY=infix, STRUCT=nonempty, index=X_15:mid} :
  @s1:mid(separator ^
          <Left-Arg>(a1:formobj ^ a1) ^
          <Right-Arg>(i1:infix-rel ^ inrel1 ^
                      <Left-Arg>(a2:termobj ^ a2) ^
                      <Right-Arg>(a3:termobj ^ a3)))
Parse 3: expr{FENCE=unfenced, FIXITY=infix, STRUCT=nonempty, index=X_14:mid} :
  @s1:mid(separator ^
          <Left-Arg>(a1:termobj ^ a1) ^
          <Right-Arg>(i1:infix-rel ^ inrel1 ^
                      <Left-Arg>(a2:termobj ^ a2) ^
                      <Right-Arg>(a3:termobj ^ a3)))
Parse 4: term{FENCE=unfenced, FIXITY=infix, STRUCT=nonempty, index=X_17:mid} :
  @s1:mid(separator ^
          <Left-Arg>(a1:termobj ^ a1) ^
          <Right-Arg>(i1:modifier ^ inrel1 ^
                      <Left-Arg>(a2:termobj ^ a2) ^
                      <Right-Arg>(a3:termobj ^ a3)))
Parse 5: form<23>{FENCE=unfenced, FIXITY=infix, GLUE=noglue, index=X0_38:infix-rel, ontology=infix-rel} :
  @i1:infix-rel(inrel1 ^
                <Left-Arg>(s1:mid ^ separator ^
                           <Left-Arg>(a1:termobj ^ a1) ^
                           <Right-Arg>(a2:termobj ^ a2)) ^
                <Right-Arg>(a3:termobj ^ a3))
```

C.5 Normalized XML of OpenCCG Logical Forms

```
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML"
        xmlns:cdlf="http://www.kwarc.info/projects/lamapun/cdlf">
  <m:apply type="mid">
    <cdlf:token type="mid">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="termobj">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="termobj">
      <cdlf:token type="termobj">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <cdlf:token type="termobj">a2</cdlf:token>
  <m:apply type="mid">
    <cdlf:token type="mid">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="formobj">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="formobj">
      <cdlf:token type="formobj">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <m:apply type="mid">
    <cdlf:token type="mid">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="termobj">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="modifier">
      <cdlf:token type="modifier">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <m:apply type="modifier">
    <cdlf:token type="modifier">inrel1</cdlf:token>
    <m:apply mode="Left-Arg" type="mid">
      <cdlf:token type="mid">separator</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj">a1</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj">a2</cdlf:token>
    </m:apply>
    <cdlf:token mode="Right-Arg" type="termobj">A1</cdlf:token>
  </m:apply>
  <m:apply type="termobj">
    <cdlf:token type="termobj">inrel1</cdlf:token>
    <m:apply mode="Left-Arg" type="mid">
      <cdlf:token type="mid">separator</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj">a1</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj">a2</cdlf:token>
    </m:apply>
    <cdlf:token mode="Right-Arg" type="termobj">A1</cdlf:token>
  </m:apply>
</m:math>
```

C.6 Established Equivalence Classes

Equivalence is established by considering all features, except the "mode" attiribute.

```
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML"
        xmlns:cdlf="http://www.kwarc.info/projects/lamapun/cdlf" baseclass="13" equivclass="18">
  <m:apply type="mid" baseclass="9" equivclass="10">
    <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="termobj" baseclass="2" equivclass="2">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="termobj" baseclass="10" equivclass="9">
      <cdlf:token type="termobj" baseclass="3" equivclass="3">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <m:apply type="mid" baseclass="9" equivclass="12">
    <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="formobj" baseclass="6" equivclass="6">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="formobj" baseclass="11" equivclass="11">
      <cdlf:token type="formobj" baseclass="7" equivclass="7">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <m:apply type="mid" baseclass="9" equivclass="14">
    <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="termobj" baseclass="2" equivclass="2">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="modifier" baseclass="12" equivclass="13">
      <cdlf:token type="modifier" baseclass="8" equivclass="8">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <m:apply type="modifier" baseclass="12" equivclass="16">
    <cdlf:token type="modifier" baseclass="8" equivclass="8">inrel1</cdlf:token>
    <m:apply mode="Left-Arg" type="mid" baseclass="9" equivclass="15">
      <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="2" equivclass="2">a1</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
    </m:apply>
    <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
  </m:apply>
  <m:apply type="termobj" baseclass="10" equivclass="17">
    <cdlf:token type="termobj" baseclass="3" equivclass="3">inrel1</cdlf:token>
    <m:apply mode="Left-Arg" type="mid" baseclass="9" equivclass="15">
      <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="2" equivclass="2">a1</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
    </m:apply>
    <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
  </m:apply>
</m:math>
```

C.7 Compact Disjunctive Logical Form

```
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML"
        xmlns:cdlf="http://www.kwarc.info/projects/lamapun/cdlf" baseclass="13" equivclass="19">
  <m:apply type="mid" baseclass="9" equivclass="10">
    <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="termobj" baseclass="2" equivclass="2">a1</cdlf:token>
    <cdlf:set baseclass="14" equivclass="20">
      <m:apply mode="Right-Arg" type="termobj" baseclass="10" equivclass="9">
        <cdlf:token type="termobj" baseclass="3" equivclass="3">inrel1</cdlf:token>
        <cdlf:token mode="Left-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
        <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
      </m:apply>
      <m:apply mode="Right-Arg" type="modifier" baseclass="12" equivclass="13">
        <cdlf:token type="modifier" baseclass="8" equivclass="8">inrel1</cdlf:token>
        <cdlf:token mode="Left-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
        <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
      </m:apply>
    </cdlf:set>
  </m:apply>
  <m:apply type="mid" baseclass="9" equivclass="12">
    <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
    <cdlf:token mode="Left-Arg" type="formobj" baseclass="6" equivclass="6">a1</cdlf:token>
    <m:apply mode="Right-Arg" type="formobj" baseclass="11" equivclass="11">
      <cdlf:token type="formobj" baseclass="7" equivclass="7">inrel1</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
    </m:apply>
  </m:apply>
  <m:apply type="modifier" baseclass="12" equivclass="21">
    <cdlf:set baseclass="14" equivclass="22">
      <cdlf:token type="modifier" baseclass="8" equivclass="8">inrel1</cdlf:token>
      <cdlf:token type="termobj" baseclass="3" equivclass="3">inrel1</cdlf:token>
    </cdlf:set>
    <m:apply mode="Left-Arg" type="mid" baseclass="9" equivclass="15">
      <cdlf:token type="mid" baseclass="1" equivclass="1">separator</cdlf:token>
      <cdlf:token mode="Left-Arg" type="termobj" baseclass="2" equivclass="2">a1</cdlf:token>
      <cdlf:token mode="Right-Arg" type="termobj" baseclass="4" equivclass="4">a2</cdlf:token>
    </m:apply>
    <cdlf:token mode="Right-Arg" type="termobj" baseclass="5" equivclass="5">A1</cdlf:token>
  </m:apply>
</m:math>
```

C.8 Final CDLF with Sharing

```
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML"
        xmlns:cdlf="http://www.kwarc.info/projects/lamapun/cdlf">
  <m:apply type="mid">
    <cdlf:token type="mid" id="equivclass1">separator</cdlf:token>
    <cdlf:token type="termobj" id="equivclass2">a1</cdlf:token>
    <cdlf:set>
      <m:apply type="termobj">
        <cdlf:token type="termobj" id="equivclass3">inrel1</cdlf:token>
        <cdlf:token type="termobj" id="equivclass4">a2</cdlf:token>
        <cdlf:token type="termobj" id="equivclass5">A1</cdlf:token>
      </m:apply>
      <m:apply type="modifier">
        <cdlf:token type="modifier">inrel1</cdlf:token>
        <m:share href="#equivclass4"/>
        <m:share href="#equivclass5"/>
      </m:apply>
    </cdlf:set>
  </m:apply>
  <m:apply type="mid">
    <m:share href="#equivclass1"/>
    <cdlf:token type="formobj">a1</cdlf:token>
    <m:apply type="formobj">
      <cdlf:token type="formobj">inrel1</cdlf:token>
      <m:share href="#equivclass4"/>
      <m:share href="#equivclass5"/>
    </m:apply>
  </m:apply>
  <m:apply type="modifier">
    <cdlf:set>
      <cdlf:token type="modifier">inrel1</cdlf:token>
      <m:share href="#equivclass3"/>
    </cdlf:set>
    <m:apply type="mid">
      <m:share href="#equivclass1"/>
      <m:share href="#equivclass2"/>
      <m:share href="#equivclass4"/>
    </m:apply>
    <m:share href="#equivclass5"/>
  </m:apply>
</m:math>
```