THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

CONTENT MARKUP LANGUAGE DESIGN PRINCIPLES

By

ANDREAS STROTMANN

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Spring Semester, 2003

The members of the Committee approve the dissertation of Andreas Strotmann defended on April 3, 2003.

Ladislav J. Kohout
Professor Directing Thesis

Mika Seppälä
Outside Committee Member

Robert A. van Engelen
Committee Member

Kyle Gallivan
Committee Member

Hilbert Levitz
Committee Member

Approved:

Sudhir Aggarwal, Chair
Department of Computer Science

The Office of Graduate Studies has verified and approved the above named committee members.

ACKNOWLEDGEMENTS

First and foremost, I wish to thank Ladislav J. Kohout without whose understanding, trust, and guidance this dissertation would have remained forever unwritten, Mika Seppälä, whose unwavering trust and support gave me so many opportunities over so many years to gain the necessary experience for this project, and the other members of my committee, R. van Engelen, K. Gallivan, and H. Levitz, for their interest, encouragement, and support.

R. Esser, F.-W. Hehl, and A.C. Hearn introduced me to the theory and practice of symbolic computing in general, and of computer algebra in particular, while J. Lenerz, P.-O. Samuelsdorff and the many participants of the Arbeitskreis Linguistik at the Institut für Deutsche Sprache und Literatur at Universität zu Köln introduced me to the fascinating research on formal linguistics of human languages. Without them, this dissertation would have looked quite different.

D. Krekel, R. Schrader, B. Haas and the members of the Graduiertenkolleg Scientific Computing at the University of Cologne got me started on the long journey which culminated in this dissertation.

Numerous members of the OpenMath working group provided invaluable opportunities to discuss and share ideas and experience in the field, especially J.A. Abbott, O. Caprotti, D. Carlisle, S. Dalmas, J. Davenport, M. Gaëtano, M. Kohlhase, A. van Leeuwen, B. Miller, M. Roelofs, M. Seppälä, R. Sutor, and S. Vorkoetter. The OpenMath Steering Committee, especially M. Seppälä, G. Gonnet, and A. Cohen, provided invaluable support, both moral and financial.

P. Marti and M. Rueher were great hosts and collaborators at ESSI/ISSS, University of Nice at Sophia-Antipolis until I was forced to stop working on this topic due to circumstances beyond their or my control in 1995. I still owe them an apology.

I wish to thank the faculty and fellow students at the Department of Computer Science, at the Supercomputer Computing Research Institute and later the School of

Computational Science and Information Technology, at the Mathematics Department, and at the Oceanography Department of The Florida State University. Special thanks also go to the people at the FSU International Student Center, especially its director, R. Christie, and the numerous members of its International Coffee Hour: they have been a haven of sanity in Tallahassee.

A.C. Hearn, W. Neun, H. Melenk, A.C. Norman, and J.P. Fitch generously provided me access to the source code of their respective implementations of REDUCE and the underlying LISP systems. S. Vorkoetter of Waterloo Maple kindly allowed me the use of his implementation of his OpenMath prototype language component in Maple. Ph. Marti provided crucial parts of our joint implementation of the OpenMath prototype language in Prolog III and Delphia Prolog. R. van Engelen provided the source code and advice on his Ctadel system as well as the funding for my implementation of OpenMath 1.0 in Ctadel.

There have been a multitude of sources of funding for the research reported upon here. The Deutsche Forschungsgemeinschaft (DFG) through its Graduiertenkolleg Scientific Computing at the University of Cologne, Germany, provided funding in the form of a PhD fellowship for the first phase of this project, and the Regional Computer Center at the Universtity of Cologne provided the use of their facilities and travel funding.

The European Community projects Editing and Computing (later renamed OpenMath) provided travel funding, as did the Research Institute for Applications of Computer Algebra (RIACA), the Symbolic Computation Group at the University of Waterloo, and, at The Florida State University, the Department of Mathematics, the Department of Computer Science, the School of Computational Science and Information Technology, the Oceanography Department, and the Research Foundation.

The Florida State University provided funding through a University Fellowship, through teaching assistantships at the Department of Computer Science, and through research assistantships at the Mathematics Department (in collaboration with the Office for Distributed and Distance Learning) and at the Oceanography Department (through a grant from the FSU Research Foundation and in collaboration with the School of Computational Science and Information Technology).

Finally, my personal and very special thanks go to D. Zhao for going through this together with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The research goal of this dissertation is to point the way towards a better understanding for the hidden complexities of knowledge communication and content markup, with a view to cleaner, more principled designs. The following four specific contributions are shown to be particularly useful tools in the quest for understanding and improving the design of content markup languages:

Linguistics parallel: Since human language is a high-quality existing solution to a generalization of the problem that knowledge communication languages aim to solve, the study of the "engineering solutions" of human language provide a guideline to engineering solutions for content markup language design.

Language layers and components: We propose a general architecture for knowledge communication languages, noting that human language appears to be structured in a deeply related parallel fashion.

Compositionality: The Compositionality Principle is a fundamental research paradigm in the study of the semantics of human language. We show how this principle, which we have introduced into our research field from linguistics, has already had a notable effect on improving content markup languages.

Categorial semantics: A categorial semantics of a knowledge communication language is another fundamental tool, as linguists who study the semantics of human language have discovered. In particular, we introduce the concept of categorial types into the discussion, and propose a complete categorial type system for one particular language, namely OpenMath.

This dissertation is thus concerned with the architecture of such languages. The linguistics parallel posits a parallel between human language and content markup architectures based on a realization that the problems they solve are deeply related, which leads us to

propose a general architecture of layers and components for content markup and knowledge communication languages. The compositionality principle provides an architectural guideline for the design of the two core layers of such languages, and writing a categorial type system for a compositional content markup language turns out to be an immensely useful tool for designing such a language.

Application of this approach to several existing language proposals provides evidence for its practical usefulness, by showing how failure to adhere to these design principles produces concrete bugs in their specifications.

CHAPTER 1

OVERVIEW

This dissertation is the culmination of a personal odyssey of almost a decade through a fascinating research area, an odyssey that began in the summer of 1993 as dissertation research on information exchange between computer algebra (CA) systems within a context of heterogeneous distributed scientific computing. We have been privileged to be a major contributor to the OpenMath development effort, especially in its formative years 1994 and 1995. Our involvement in OpenMath and MathML unfortunately had to be put on a back burner after that, but our dissertation project was then finally revived in early 1998 with a focus on investigating the design principles of knowledge communication languages (or, content markup languages) under Ladislav J. Kohout. Chapters 3 and 5 include an account of some aspects of this intellectual journey.

During this time, but especially since 1998, we have published on our research in several papers [1, 2, 3, 4, 5], and presented at several more international workshops in the field. Our research has been acknowledged to have contributed significantly to the development of two particularly important content markup languages in the mathematics domain, namely OpenMath [6, 7, 8, 9, 1, 10] and MathML [11, 12, 13, 14, 15, 16].

The title of this dissertation, "Content Markup Language Design Principles," is unfortunately far too ambitious for us to be able to work out in full detail the full spectrum of topics that it implies. A full exploration of the design principles of content communication languages will of necessity be a long research expedition that this dissertation is only the initial exploratory foray for. Therefore, the more manageable task of this dissertation is to motivate one particular approach (the linguistics parallel) that promises to guide us on the way, and all we can reasonably expect to do within the constraints of a single dissertation is to visit a few milestones on that road to show that it is leading in the right direction, and

to venture some predictions on where else this road is going lead us, and the lessons we may come to learn along the way.

Thus we hope to help gain a better understanding of the hidden complexities of the problem of communicating knowledge among software system, as a significant contribution to the scientific study of this problem.

1.1 The Problem Area

To summarize very briefly Chapter 2, which explains the research area at length, our problem area is concerned with communicating computer-understandable symbolic knowledge in a network of heterogeneous knowledge-based software systems.

Less generally, we have studied the design and implementation of common languages for communicating mathematical knowledge between different mathematical software systems, or languages for representing such knowledge on the World Wide Web. In particular, I have made contributions to the design and specification of the OpenMath and MathML standards that have been publicly acknowledged by the relevant bodies.

1.1.1 A Deceptively Simple Problem

During the 1994 OpenMath Workshop [17] in Oxford, G. Gonnet expressed the hope that a stable OpenMath version 1.0 would be published within a year or two. Simultaneously, D. Raggett was leading a small group of enthusiasts in an attempt to include a representation language for mathematical formulas in the then upcoming version of HTML.

By contrast, the Text Encoding Initiative (TEI) published a recommendation [18] for document markup in SGML at roughly the same time which explicitly excluded the handling of mathematical formulas from its scope because it was felt that it needed much more research than the TEI were able to finish within the time frame allotted for the project.

And indeed, both the OpenMath and the HTML-Math projects turned out to be grossly over-optimistic in their time-plans (as indeed was our own planned schedule for this dissertation, which began about the same time). MathML 1.0 was finally published in mid-1998 [13], with many corrections both to its Content and its Presentation Markup requiring the publication of MathML 2.0 [16] as a first stable version in early 2001, and

OpenMath 1.0 was finally published in early 1999 with a stable official version of a core set of "Content Dictionaries" completing it in 2001.

Eight years of intense effort were thus required instead of the two years or so originally envisaged by leaders of the field. The problem was much more difficult than originally thought – it looked deceptively simple. And even now, as we will see, solutions that have been proposed have been problematic in important ways.

Similarly, another knowledge communication language that we will analyze in the course of this dissertation, namely the Knowledge Interchange Format (KIF) language, has been going through more than a decade of intense development within the symbolic artificial intelligence community, with several versions each introducing major changes in the language over the years.

1.1.2 Problems of Quick-and-Dirty Solutions

It is interesting to contemplate why the problem of defining a mathematical content communication language for computer algebra systems (in the case of OpenMath) or a mathematical content markup language for the web (in the case of MathML) was so gravely underestimated.

In both cases, major system vendors for computer algebra systems (CASs) – Axiom, Maple, Mathematica, REDUCE and others – were heavily involved. In a sense, they all believed they had already solved a problem just like this one: there is a clear distinction between the user-level language and the system internal representation of mathematical formulas in any and all of these systems, and the problem appeared to be simply to come up with a compromise between all of the user-level languages "spoken" by these systems. And such a compromise was indeed easy to come up with – S. Vorkoetter's 1994 draft OpenMath proposal [19] was one, with a "natural" LISP-like syntax, written down within just a few weeks in early 1994.

However, as the OpenMath project at least progressed, it began to dawn on the participants that there were a few differences between those problems that CASs had previously attacked (and presumably solved), and those that were posed here, among them:

- All existing languages, being essentially programming languages, had procedural semantics; by contrast, OpenMath, as a communication language, needed a declarative semantics.

- However, defining a semantics for CAS languages is not actually well understood. The interplay between functional, procedural, and rewrite-rule processing styles within a single systems is very tricky, for example. Type systems for mathematics on the one hand and object-oriented type systems as found in software systems do not match well, either [20].

- Perhaps as a consequence, both syntax and semantics in CASs is mostly handled in a very ad-hoc fashion, while data structures and operations on them tend to be highly tuned. Unfortunately, at the first level of approximation, a language like OpenMath is defined mostly in the former, and less in the latter, realm. Indeed, both MathML and OpenMath in their current incarnation ignore the data-structure component of communication almost completely.

Thus, even though it was fairly easy to come up with a quick solution that would allow the major CASs to exchange the mathematical meaning of run-of-the-mill formulas, such easy solutions were also dirty: they did not scale well, and they were error-prone, as we have argued in [2] and as we will show in this dissertation. In a nutshell, they didn't solve the problem they were meant to solve, if that problem was defined as helping tie together a class of software systems that went well beyond the few major CASs, and as being able to communicate mathematical knowledge about a steadily growing and more and more complex class of concepts.

1.2 A Better Understanding for More Principled Solutions

A better understanding for the hidden complexities of the problem of content communication and content markup with a view to cleaner, more principled solutions – that, in a nutshell, is then the *main contribution* that I hope and claim to make to the scientific study in this field in this dissertation.

The following four *specific contributions* in particular have shown themselves to be useful tools in the quest for understanding and improving the design of content communication and markup languages:

Linguistics parallel: In [7, 1, 2, 21] we outlined a view that, since human language very likely is a high-quality existing solution to a more general version of the problem that content communication and content markup languages aim to solve, the study of the "engineering solutions" [22] that human language contains should point to good engineering solutions for our problem. Chapter 6 argues for this approach in considerable detail.

Language layers and components: In particular, we described an analysis of the layers and components that need to make up a content communication language in [7, 1], noting that human language appears to be structured in a parallel fashion.

Compositionality: The *Compositionality Principle* is a fundamental research paradigm in the study of formalizations of the semantics of human language [23]. This principle, which we introduced into the OpenMath and MathML discussions, has had a notable effect on improving the language design in both cases. Chapter 7 is devoted to a discussion of this principle and its application and importance in knowledge communication languages.

Categorial semantics: The concept of a *Categorial semantics* is another fundamental ingredient in the toolbox of linguists studying the formal semantics of human language. In Chapter 8, we apply one of the tools comprised under this heading, *categorial type logics* [24], to the OpenMath language, with surprising and important results that have led to at least one major revision of the OpenMath core language.

Our contributions can thus be classified as concerning the *architecture* of content markup or communication languages:

- The linguistics parallel posits an architectural parallel between human language and content markup based on a realization that the problems they solve are deeply related to each other: "There are powerful analogies between studying, say, human conversation and information exchange between machines,"as [25] put it.

- The layers and components of content communication languages that we derive from the linguistics parallel prescribe a specific architectural decomposition for such languages along one particular dimension.
- The compositionality principle has proven its value as an architectural guideline for the design of content markup languages.
- The concept of categorial semantics in general is another aspect of the architecture we propose for content communication languages, while the exercise of writing a specific categorial type system for such a language is a very useful tool in the actual design of such a language.

1.2.1 The Linguistics Parallel

In [7, 1, 2, 21] we outlined a view that, since human language very likely is a high-quality existing solution to a more general version of the problem that content communication and content markup languages aim to solve, the study of the "engineering solutions" [22] that human language has come up with should be able to point to good engineering solutions for our problem.

First of all, this approach leads us to propose a very high-level architecture for solutions to our problem, that is a way to divide up the problem into "natural" layers and components – syntax, semantics, pragmatics, as well as lexicon and other modules (see also section 6.2).

Based on this "natural" architecture, the rest of chapter 6 gives examples of more or less specific solutions inspired by this "linguistics parallel" in each of the components and layers we propose for content markup languages. Some of these are simply motivational in nature, but the remaining three of the main contributions to the field that I claim to make in this dissertation, which have been worked out in considerable detail and which have proven their worth with respect to the design of content communication languages, are meant to serve as convincing evidence that the "linguistics parallel" is an important and surprisingly fruitful and practical approach to understanding and improving content markup language design.

Thus, the claim we make in this part of the dissertation is not a purely technical one but rather a philosophical one, claimed to be valid to a certain degree rather than to be true absolutely. It is the degree to which the following three examples both are parallel

in an important and non-trivial fashion to aspects of linguistics and have important and non-trivial consequences for the design of content markup and communication languages, that determines the degree to which this claim is valid and important.

Beyond this, the more "motivational" examples that have not been worked out to the degree that the three described below have, can serve to establish the importance of this approach as a framework for future research on further aspects of content markup languages that are only sketched, but not fully developed yet in this dissertation.

1.2.2 Content Markup Language Architecture

In [7, 1] we described an analysis of the layers and components that need to make up a content communication language, noting that human language appears to be structured in a parallel fashion. Indeed, we note that the latter provided the intuition for the former. Since OpenMath did not finally follow the concrete layering we proposed in that paper, it is instructive to see the problems that have surfaced in that language as a result of the decision to use a simplified structure. Similarly, other language proposals, but also reports on implementations of such languages can be fruitfully analyzed in terms of the structure we first proposed at the Third OpenMath Workshop in Amsterdam in February 1995 [26].

1.2.3 The Compositionality Principle

The Compositionality Principle is a fundamental research paradigm in the study of formalizations of the semantics of human language – so much so that an entire chapter of the 1997 "Handbook of Logic and Language" [25] is titled "Compositionality" [23].

We first introduced the Compositionality Principle of language design into the OpenMath discussion at the 1996[1] OpenMath Workshop in Dublin [27] during a long and controversial discussion on the representation of integration (as a representative of the class of first- and

[1]Note that this pre-dates the publication of the "Handbook of Logic and Language" [25]. Nevertheless, the Handbook is an appropriate reference to give in this context because it contains excellent reviews by many of the same authors on many of the same topics as those of the 1988 Workshop on *Computational Linguistics and Formal Semantics* held at Istituto Dalle Molle in Lugano, Switzerland, where we first became acquainted with its topics.

higher-order operators; see also [3] for a discussion), and into the MathML discussion in a posting to its public `www-math` mailing-list again in 1997.[2]

As we discuss briefly in [2] and later in more detail in [3], this principle has led to some important and unusual design decisions for OpenMath, especially for representations of first- or higher-order operators. We have analyzed several knowledge communication or content markup languages with respect to the compositionality especially of their representation of such constructs, and located serious bugs in several of them, some of which appear to have gone unnoticed for years (op.cit. and [28]).

In addition, the compositionality design principle is a necessary prerequisite for the categorial types design tool described in the next section, which has flushed out (and fixed) yet more language bugs that had remained unnoticed for years.

1.2.4 Categorial Semantics

Categorial semantics of language is another fundamental ingredient that we discovered in the toolbox of linguists studying the semantics of human language, a tool that again is important enough in that field that one of its components serves as the title of another chapter in the *Handbook of Logic and Language* [25], namely *Categorial Type Logics* [24].

We have introduced this concept into the discussion on the design of content markup languages in two ways. First, it is an important motivation for proposing one of the two central layers of our OpenMath language model in [7, 1], namely the layer we called the OpenMath *Expression* layer. Second, we have successfully applied one of the tools comprised under the heading of categorial semantics to OpenMath, proposing a *Categorial Type System* as a candidate for extending the OpenMath standard [29, 30].

In addition to providing a valuable tool for specifying cleanly the semantics for OpenMath 1.0 [10] independently of its add-on *Content Dictionaries* by providing a novel and unified interpretation of the OpenMath Objects it defines, we again flushed out some subtle problems both in the OpenMath 1.0 definition and in two previous attempts at defining a semantics for OpenMath 1.0 [31, 32].

[2]Note that I have since learned that compositionality is an important design principle in many fields of computer science. However, as we have pointed out, its applicability in the particular case of content communication is more deeply rooted than its practical usefulness in other fields.

In summary, the latter two of our main contributions to the set of tools for designing content markup and communication languages, the Compositionality Principle and Categorial Types, strongly argue for the more complex Layers and Components structure of such languages that we proposed as a third important argument, which in turn together with the other two provides a strong argument for the importance and a high degree of validity of the Linguistics Parallel approach to the problem of designing content markup languages that we propose as our first main contribution to the field.

CHAPTER 2

INTRODUCTION

The title of this dissertation is *Content Markup Language Design Principles*. Few readers will have an impression of the intended meaning of that phrase, or indeed of the setting within which it is meant to be read. For this reason, this chapter introduces the audience to the area of research that provides the backdrop for this study, and explains how this dissertation fits into that research area.

Chapters 3 and 4 will then review the literature in the research area of this dissertation. Each chapter reviews the field from a different perspective.

Chapter 3 takes an in-depth look at the development of just a few central content communication and content markup languages, mostly within the field of Computer Algebra (CA). Since we have been deeply involved in the two main threads of this development, it is written from a very personal perspective, hopefully giving the reader a flavor of this exciting and ongoing research field.

Chapter 4, on the other hand, discusses related topics, including areas of application for content markup languages that have been explored in the literature. It classifies the literature into research topics and surveys each briefly.

Chapter 5 then rounds out the practical perspective on our topic. It reports on the practical experience we gained in several projects over the years in which we implemented content communication in order to link a variety of symbolic and numerical computing systems in order to build specific cooperative and distributed problem solving systems. This chapter motivates the more theoretical investigations in later chapters by explaining how the intuitions that lead to them are grounded in practical experience.

The rest of the dissertation investigates the topic of content markup language design in depth. The focus in these chapters is on the underlying theory, but this is balanced by concrete applications of the theory to concrete language proposals in the literature.

In Chapter 6, we give a detailed description of our main approach, namely the "powerful analogies" that we perceive in this particular case of designing knowledge communication languages "between studying human conversation and information exchange between machines," as [25] put it in another context. Since human language is likely a high-quality existing solution to a more general version of the problem that content communication and content markup languages aim to solve, we argue here that the study of the "engineering solutions" [22] that the human language faculty contains should point to good designs for solutions to our problem. In particular, we described an analysis of the layers and components that need to make up a content communication language in [7, 1], noting that human language appears to be structured in a parallel fashion.

The *Compositionality Principle* is a fundamental research paradigm in the study of formalizations of the semantics of human language [23]. This principle, which we introduced into the OpenMath and MathML discussions, has had a notable effect on improving the language design in both cases. Chapter 7 is devoted to a discussion of this principle and its application and importance in knowledge communication languages.

The concept of a *Categorial semantics* is another fundamental ingredient in the toolbox of linguists studying the formal semantics of human language. In Chapter 8, we apply one of the tools comprised under this heading, *categorial type logics* [24], to the OpenMath language.

Finally, Chapter 9 summarizes again the main results of this dissertation, and provides an outlook to new questions that it raises for future research.

2.1 The Research Area

This dissertation is situated in the research area of content markup and knowledge communication languages.

This field is a new multidisciplinary area of research rooted in several related fields and sciences. In Computer Science, the research areas of Artificial Intelligence (Agent Communication, Knowledge Communication, Intelligent Systems Design) and Networking (WWW, XML, Communication languages) are particularly relevant, as are recent developments in the design and implementation of Problem Solving Environments, where Computational Science plays an additional role. The field's historical roots can be traced to the design

and implementation of symbolic computation systems (e.g. Computer Algebra, Theorem Proving, Knowledge Representation). Its theoretical foundations come from the cognitive sciences and linguistics, in particularly formal logics and formal semantics such as Fuzzy Logic and Fuzzy Systems.

Application areas of this field of study are abundant. The prototypical application area has been the design and implementation of content markup languages for representing the semantic content (meaning) of mathematical formulas for communication amongst symbolic computation systems or in web documents. Since 1994, we have made major contributions to a seminal effort in this application area, the OpenMath project [19, 8, 1, 10], where we co-authored the official requirements and problem analysis document for that international project of the Computer Algebra community, titled *OpenMath Objectives* [6, 7, 9, 1].

The work of this group carried over into that of the World Wide Web Consortium's Working Group on Mathematics, especially into its Content Markup subdivision the members of which have mostly simultaneously been members of the OpenMath group. The specifications for a Mathematical Markup Language (MathML) acknowledge both our specific direct contributions and the influence of the OpenMath effort.

Beyond mathematics, all the sciences, indeed, all the liberal arts, are application areas for Content Markup Languages that allow structured knowledge from their respective fields to be communicated across computer networks and between a wide and diverse variety of programs and systems.

In the following, I would like to explain in more elementary terms the extent of this research area, by explaining the several terms comprising the title of my research topic.

2.2 Markup

The term "markup" originates in the publishing business, where it originally meant the marks that editors, lectors, and printers add to drafts of texts to communicate among themselves about the changes they require to be made to the layout or the structure of a text that is going through the editing process.

In computer applications, text markup has acquired the generalized meaning of information added to a text (i.e., information that does not appear in print) which describes the logical structure of the text (chapters, sections), defines aspects of the layout of the

text (bold-face, page break), or which contains lightly structured information about the text (author, date published, ISBN). These three types of markup have been known in the text markup literature as structural markup, specific markup, and meta data, respectively.

The ISO standard SGML (Standard Generalized Markup Language, [33]) was designed for use by publishers to replace their paper-and-pencil based document designs and document markings within a computerized environment. SGML allowed the definition of extremely complex Document Types (DTDs) that specify the logical structures available for documents of its type and their syntactic representation in these documents; such DTDs would then be compiled into parsers and processors for text documents of the specified type.

Due to its staggering complexity, SGML never became widely used outside big publishing houses, despite such grandiose efforts as those of the Text Encoding Initiative [18]. Instead, a few simple DTDs eventually became widely used without explicit reference to their SGML DTD as the World Wide Web began its explosive development over the last decade.

Thus, the term "Markup" as used in the heading of this section of the paper is best known today as the "M" in "HTML" – the hypertext markup language [34]. Few people have ever bothered to look at actual DTDs for HTML, even though HTML was meant to work as an SGML application. Instead, the very simple syntax defined in the HTML document type was typically written and interpreted directly without the help of a generic SGML tool.

Nevertheless, HTML is today "the" prototypical example for the term "Markup" as it offers descriptors for the logical structuring of documents (headers, paragraphs, cross-references), for layout control (bold face, tables), and even for marking editorial changes (strike through).

A more recent addition to HTML is actually an integral part of the concept of text markup: style sheets [35] that map the logical structure of a document to a screen or paper layout that visualizes that structure to a reader. This is an important development as it helps separate the logical structure from its realization in the form of specific printing conventions.

2.3 Markup Language

Thus, a Markup Language comes in several parts: a language that is used when defining a document type (that is, for writing DTDs[1]), a language for the markup that encodes the logical structure of a document according to such a DTD, and a way to intermix the markup with the plain text of a document while retaining the ability to distinguish between markup and text.

Recently, there has been a huge effort in the software and Internet industry to develop a simplified SGML for use on the Web, dubbed XML for "eXtensible Markup Language" [37]. XML provides the basic syntactic framework for a whole family of more or less closely related languages. Core XML provides the basic syntactic features for container or empty elements with attributes and attribute values, Unicode text and entities, and comments, DTDs, and processing instructions.

The XML effort includes a large number of ongoing projects to add more standard functionality and usefulness to the basic syntactic framework of XML, such as style sheets (XSL [38]), cross-referencing (XLink[39], XPointer[40]), name spaces [41], embedding of XML fragments in other XML documents (XML Schemas [36]), to name just a few. Much of this is being done in the form of projects of the World Wide Web Consortium in conjunction with the software industry.

2.4 Content Markup

The term "content markup" traces back to the literature on text markup, where it appears to have been used routinely for a long time. In this field, it was used to describe the kind of text markup that needed the text specialist's understanding to both provide and utilize. This is in contrast to "structural" text markup such as "title", "footnote" and the like that can be entered into an electronic version of a text without a need for the typist to understand any of the text being entered. Thus, it was used to describe markup that required understanding of the content, and not just the form, of the text being marked up.

[1] More recently, efforts are underway to define document types by sets of "Schemas" [36], but the underlying principle is the same.

In the history of HTML, the term "content markup" can be traced back at least to discussions on HTML 3,[2] but with a slight shift of meaning. On the one hand, it is related to the text markup terminology in that it distinguishes markup for "content" from markup for "form" in the sense of units of meaning versus units of layout, but on the other hand the term tends to be generalized to covering the "structural" markup from the text markup field, since its main function appears to be to distinguish "title" and "header" content markup from "font" and "bold" presentation markup. Content markup is now used to denote such markup as can be "understood" (and manipulated) by the computer rather than by the human specialist as in the text markup field.

It was in a combination of these two senses that the term "Content Markup" was used by the writers and editors of the World Wide Web Consortium's "Mathematical Markup Language" Recommendations [14] to distinguish between those parts of MathML that were developed to convey information about the two-dimensional layout of a mathematical formula ("presentation markup") and those parts of MathML that were designed to convey the meaning of a formula irrespective of the details of its layout ("content markup"). This sense of "content markup" is closely related to the one found in the field of text markup, since an understanding of a mathematical formula is necessary both to produce and to utilize its content markup in the MathML sense. At the same time, MathML emphasizes that its content markup is designed to enable automatic processing by software such as computer algebra systems, which brings this terminology in line with the way it was historically used in the HTML field.

To see the need for the distinction between content and presentation markup for mathematical formulas, consider the character "e" in the following two example expressions:

$$a + bx + cx^2 + dx^3 + ex^4$$

$$e^x$$

Clearly, even though the two "e"s are printed identically, the knowledgeable reader might interpret the one in the first formula as representing an arbitrary constant (of some unspecified domain) given a context in which polynomials can be expected, but they might

[2]There are numerous contributions to the *www-math* mailing list going back at least to 1994/1995 that defend a view of HTML as a "content markup language." See also [42].

recognize the character "e" in the second expression to stand for a particular real constant, namely the base of the natural logarithm, in a calculus context.

Thus, identical presentation elements may have wildly different meanings in a mathematical formula, and it is arguable that automatic processing of a formula with mathematical software would require a disambiguated content markup, because disambiguation of presentation markup requires understanding of the content and in general would require too much intelligence to be left entirely to the computer to figure out.

In our example, therefore, the goal of Content Markup is to provide the means to encode a mathematical formula or some other piece of visible text in such a way that its meaning is clarified: the second "e" would be content-marked-up as "the base of the natural logarithm" (and rendered as an "e" by applications in which the base of the natural logarithm is commonly printed as a lower-case letter "e"), while the first would be marked up as a "parameter", perhaps (if known) together with the range of possible values for that parameter.

2.5 The Mathematical Markup Language: Content and Presentation Markup for Mathematics

As mentioned above, the World Wide Web Consortium has published official Recommendations for marking up mathematical formulas on the WWW, called MathML (or Mathematical Markup Language [14, 15]). MathML was developed in parallel with, and as the prototypical "worst-case" application of, the Consortium's XML Recommendation [37].

The MathML Recommendation is subdivided into MathML-Content and MathML-Presentation. Of these two, MathML-Content is content markup that covers a fixed, basic set of mathematical concepts from K-12 mathematics education. A mechanism is also provided in MathML-Content to refer to external definitions of mathematical concepts; in particular, OpenMath [10] is a designated such extension mechanism for MathML to higher mathematics.

Here is an example for MathML content markup, a representation for e^x:

```
<apply>
  <power/>
  <exponentiale/>
  <ci>x</ci>
</apply>
```

This expression is easily interpreted as the "application of the powering operator to the base of the natural logarithm (*a.k.a.* exponential 'e') as base and the identifier 'x' as exponent", in short, "e to the x". Note the use of the primitive concepts of "application" and "identifiers" (functions, predicates, quantifiers, and type constructors may be "applied" to their arguments), and the representation of the operator for powering and the constant "exponential e" by empty content-markup elements.

Another way to express this concept in Content MathML is:

```
<apply>
  <exp/>
  <ci>x</ci>
</apply>
```

This expression represents the application of the exponential function to its one argument, the identifier 'x.' Again, a mathematician would know to read this as "e to the x."

2.6 Presentation Markup

Contrast this with the same expression marked up using MathML-Presentation markup:

```
<msup>
  <mi>e</mi>
  <mi>x</mi>
</msup>
```

This clearly reads "e superscript x", which may in many cases mean "e to the x", but may also stand for "the unit vector for the x axis" or any one of a number of different concepts. Note here the use of layout concepts for marking up the superscript presentation, and the

use of an "identifier" concept which actually means "printed in the font and style typically used for printing identifiers."

The following example ($e + d\,x$) shows a few more interesting features of MathML Presentation:

```
<mrow>
  <mi>e</mi>
  <mo>+</mo>
  <mrow>
    <mi>d</mi>
    <mo>&InvisibleTimes;</mo>
    <mi>x</mi>
  </mrow>
</mrow>
```

Operators and identifiers are distinguished because subtle spacing rules need to be observed when typesetting them, but conceptually both the sum and the product are simply horizontal rows of printed symbols. However, in order to support automatic line breaking, for example, such rows are nested according to their logical scoping. Note also the use of an "invisible" operator: instead of a horizontal row of "d" and "x", the recommendation (again primarily for subtle typesetting reasons) is to use such "invisible" operators for multiplication, application, or similar concepts.

2.7 Markup Language Design

Note how the two disjoint subsets of MathML, its Content and its Presentation Markup language, follow two distinct language design patterns. While content markup is unashamedly LISP-like in design – the prefix notation and the concepts of application, operators, variables, and constant symbols all evoke the LISP history –, presentation markup is essentially a left-to-right, top-to-bottom linearization of the two-dimensional layout of a formula, though with logical structuring made explicit (which latter property distinguishes MathML from TEX [43]).

It is interesting to note here that historically, the first public drafts of MathML-Content [11, 12] were much closer to the current MathML-Presentation design than to the current MathML-Content [16] language design, using infix, postfix, or prefix notations depending on the default rendering for the content elements. Clearly, there are therefore important non-obvious issues to be weighed when designing a markup language, and different design principles may easily apply in different domains or on different logical levels of the same domain.

The latter observation becomes particularly important when designing markup languages that are meant to be intermixed with markup from other domains (say, physics within a chemistry context) or to be mixed with markup from different logical levels from the same domain (such as presentation and content markup for mathematical formulas). MathML avoids the complications that would arise by allowing free inter-mixing of markup from the strictly prefix content domain with the mostly infix presentation domain, by cleanly separating the two kinds of markup and clearly specifying an interface between the two.

2.8 The Research Topic

Within the research area outlined above, our particular theoretical research interest is in investigating fundamental design principles for content markup and knowledge communication languages. The ansatz we follow here is to analyze research results about the structure and underlying design principles of human language and communication, focusing on the compositionality principle and the related concept of a categorial semantics, to apply them to the design of content markup and knowledge communication languages, and to show how a deeper understanding of these design issues improves the implementation of knowledge communication languages.

2.9 Understanding Content Markup Language Design

As we have seen, designing content markup languages is an important current activity in computer science and its applications. It may also have become apparent from the above

that this design process is currently treated mostly as an art rather than a science, with taste being a very important factor together with some practical and technological considerations.

Gaining an understanding of the design principles for content markup languages is an important goal both in and of itself (since investigating the scientific basis for a new area of technology such as content markup is important), and because of the important fields of application such as mathematics, computer science, chemistry, and biochemistry for which content markup languages are being developed. The latter would likely profit from a deeper understanding of the issues of content markup language design, helping as it would to produce higher-quality content markup languages, just as the corresponding languages for mathematics have already profited.

2.10 Towards Understanding Content Markup Language Design

Our long-term goal is therefore clear: to deepen our understanding of what it is that makes high-quality designs for content markup languages, and of what it is about them that would help or hinder exchange of knowledge encoded with them between a diversity of applications both on the Web and off.

What is less clear is which road to travel towards this goal. So far, efforts have mostly been haphazard, producing flawed designs for lack of understanding of deeper issues (see Chapter 7 for several examples).

In [1] (originally distributed as a working paper of the OpenMath community under the title of "OpenMath Objectives" [6, 9] and presented as a poster at ISSAC'95 [7]) we first proposed a promising approach to gaining an understanding of the problem of encoding of "meaning" (i.e. semantic content) in content (markup) languages. In that paper we proposed to analyze and use what has been discovered about the one "good" solution for knowledge communication known to exist: human language and communication among human beings.

Fields of science that are currently studying communication in general and languages and how they carry meaning in particular include philosophy, linguistics, mathematics, computer science, and cognitive science. Much has been discovered in recent decades both about the general structure and about the details of how human beings manage to convey their meaning to other human beings. A central discovery of the last half-century,

generally attributed to Noam Chomsky, concerns what is now known as Universal Grammar – underlying grammatical principles that can be found in all human languages (including the sign languages spontaneously developing among the deaf [22]). Since the human language capacity has almost certainly evolved under great evolutionary pressure (see e.g. [22, 44] for a summary of arguments), the universals of human communication in general, and of human languages in particular, can reasonably be assumed to be high-quality solutions to fundamental problems of content communication in general.

However, the well-known fact that human languages are extremely hard to process on a computer despite their simple and elegant structure should also serve as a caution that some of the universal aspects of language are most likely due to the evolutionary accidents and biological constraints of the human species. Thus, while taking seriously the homology between, on the one hand, the ability of human beings to communicate their meanings to others, and on the other hand, the goal of content markup to do the same among software systems, is a very promising approach towards a deeper theoretical understanding of the latter, this approach does come with an inherent potential practical pitfall; indeed, through circumnavigation of this pitfall, the most important result of research that follows through this ansatz might well eventually be a better understanding of which language universals are due to underlying fundamental communication issues, and which to biological accidents of the human species.

Thus, the basic idea of our research is to evaluate the design principles that have been discovered in recent fruitful attempts at understanding the fundamental structure of human cognition in general and of human language in particular, and to extract from these design principles those that are due to fundamental issues in the communication of meaning, while leaving out those messy details that make human languages easier for humans to understand, but impossibly difficult for computers to handle.

2.11 The Linguistics Approach

Because the theoretical linguistics approach to content markup language design, presented in this way, sounds both obvious and impractical given the history of, say, machine translation research, it is important to emphasize that it was actually through repeatedly

stumbling onto problems and concepts that looked familiar from linguistics while working on practical issues in content markup design and implementation during my involvement with the OpenMath effort[3] that it eventually dawned upon us that the linguistic approach also might be able to provide practical solutions to practical problems if considered from the right perspective.

The fact that some of these solutions are not simple, and that logicians who studied the formal semantics of language learned quite a bit of unsuspected new mathematics in the process [25], is thus presumably due to the inherent complexity of the problem they attempt to solve.

2.11.1 Compositionality

A key case in point of this approach and its extreme usefulness for the proper design of content markup languages, namely the compositionality principle [23], is discussed in Chapter 7. Through my personal effort, it has proved quite influential in the designs of both MathML and OpenMath.

The compositionality principle, which says that *the meaning of a compound expression is a function of the meanings of its parts and of the syntactic rule that glues the parts together* is an important guideline in current research into the detailed logical structure of the meaning carried by human language [23]. One of the results of this principle with respect to the structure of human language is the treatment of lexical binding by means of special syntactic constructs.

In the case of finding good ways to represent mathematical formulas, it is the treatment of variable binding operators like integration or quantification that is a particularly interesting test case where the application of the compositionality principle provided especially useful insights.

[3] See Chapter 5 for a summary of our practical experience and lessons it taught.

2.11.2 Categorial Semantics

Closely related to the Compositionality Principle, the research field of Formal Semantics in Linguistics provides us with Categorial Semantics[4] [24] as a powerful design tool for Content Markup languages.

Basically, the idea behind a "categorial semantics" is that a compositional language allows one to define its semantics by specifying, first, for each syntactic constructor in the language a corresponding function that takes the meanings of the components of a compound expression and produces the meaning of the compound, and second, a way to look up the meaning of any syntactically atomic element of the language. The study of the mathematical properties of a purely structural semantics divorced from the semantics of its atoms beyond that induced by their (syntactic) categories, is called "categorial semantics"; the study of the semantics of its atomic ingredients is known as "lexical semantics."

In this sense, the separation of the semantics of a knowledge communication language into a categorial and a lexical semantics is a practical incarnation of the compositionality principle.

As we show in Chapter 8, the definition of a categorial semantics is a very useful design tool for content markup languages, and again it is the design of constructs for representing first- or higher-order concepts where it is particular important.

2.11.3 Language Architecture: Layers and Components

Another fundamentally important contribution that the linguistics ansatz to content markup language design gives us is a "natural" divide-and-conquer attack based on the layers and components that research in linguistics generally ascribes to human language. Consequently, our earliest contribution (in [7, 1]) to the discussion in this field that was motivated by this approach was a proposal of a layered structure of OpenMath that corresponded roughly to a distinction between morpho-syntax, syntax, categorial semantics, and lexical semantics layers and components (though the terminology we used was slightly different).

[4]The term "categorial" in this context stems from the "syntactic categories" of linguistics rather than the "category theory" of mathematics.

Especially the splitting of the semantics layer into a structural and lexical part was a completely novel approach in the field, the importance of which is underscored by the way it has supported the introduction of the concepts of compositionality and categorial semantics into the field.

CHAPTER 3

HISTORICAL TIME-LINES

3.1 A Modern Prophecy

As early as 1945, Vannevar Bush wrote, with remarkable insight that holds today as it did then:

> "Ideas are beginning to appear for equation transformers, which will rearrange the relationship expressed by an equation in accordance with strict and rather advanced logic. Progress is inhibited by the exceedingly crude way in which mathematicians express their relationships. They employ a symbolism which grew like Topsy and has little consistency; a strange fact in that most logical field.
>
> "A new symbolism, probably positional, must apparently precede the reduction of mathematical transformations to machine processes. Then, on beyond the strict logic of the mathematician, lies the application of logic in everyday affairs." [45]

This quote starts with a good description of what would eventually evolve as the fields of Automatic Theorem Proving (ATP), which studies the algorithmic transformation of logical formulas, and of Computer Algebra (CA), also known as Symbolic and Algebraic Manipulation (SAM), which studies algorithmic transformations of mathematical formulas.

Bush continues with a brilliant insight into one of the major problems of this then-emerging field. Practitioners of symbolic mathematics needed to dig beneath the surface of the mathematicians' layout of formulas and express their underlying meaning instead, using "a new symbolism, probably positional" like the Lisp-like prefix styles that many of these systems would eventually follow internally.

However, the problem was actually worse than he thought, though his insight gives us an explanation: "[mathematicians] employ a symbolism that has little consistency." Unfortunately, as it would turn out eventually, many systems that would implement the kinds of algorithms he was envisaging would qualify for exactly the same characterization of "having grown like Topsy and having little consistency." Boys will be boys, and mathematicians will be mathematicians...

During the last ten years or so, new "symbolisms" have been developed both in the World Wide Web setting and in the symbolic computation worlds of artificial intelligence and computer algebra that were meant to bring some order to the chaos of languages that have been growing "like Topsy" in the fields over the last 40 years or so.

However, as we can see [2], even these newer proposals again grew "like Topsy" at first, by again failing, like the predecessors they were meant to unify, to properly investigate the actual underlying logical structure of the "exceedingly crude way in which mathematicians express their relationships" while transforming it into the "new symbolism, probably positional."

This we claim as our main contribution to the field: that we managed to convince those developing these new languages of the necessity of looking at the underlying semantic structure, and that we introduced the crucial concept of a compositional syntax and semantics from the field of linguistics (which could be used study the mathematicians' way of expressing themselves) into the discussion on proper designs for such "new symbolisms."

Our quote from V. Bush closes with a vision that is still very much a vision of the future: "Then, on beyond the strict logic of the mathematician, lies the application of logic in everyday affairs." We are convinced that the only way to get there is through the same principle that we applied in the development of the "new symbolism" for mathematics – namely by studying the actual underlying conceptual structures employed in these fields.

3.2 Recent Wish Lists

In a 1993 survey [46] of one of the fields of science that Vannevar Bush had envisioned half a century earlier, the joint special interest group in Computer Algebra of the German professional associations of physicists, mathematicians, and computer scientists named the development of standardized interfaces as one of the field's important research directions:

> "Schnittstellen zwischen verschiedenen [CA-]Systemen sind bis auf wenige Ausnahmen (Simath und Kant-V2, Cayley und Kant-V2) kaum vorhanden. Von einer Standardisierung von Ein- und Ausgaben, Datenstrukturen, Funktionsaufrufen, Graphik etc. wie es aus Anwendersicht wünschenswert wäre, ist man noch weit entfernt. Dieses wichtige Thema sollte in Zukunft auch international verstärkt diskutiert und angegangen werden." [46] [1]

Five years later[2] and again across the Atlantic, Erich Kaltofen presented his own short-list of ten open Computer Algebra problems to the international Computer Algebra community, upgrading the urgency level of the inter-communication and inter-operability problem to "crucial:"

> "*Open Problem 8:* Devise a plug–and–play and generic programming methodology for symbolic mathematical computation that is widely adopted by the experts in algorithm design, the commercial symbolic software producers, and the outsider users. From our FoxBox experience I observed that designing a system that simultaneously plugs into several others is difficult. The reverse is also true, namely, that designing a system that someone else can plug-in is difficult. However, our symbolic software is becoming very complex and a solution to Problem 8 is crucial." [47]

This may also have been the first time that one of the leading figures of the field of Computer Algebra openly acknowledged that this problem is not just crucial, but also unexpectedly difficult. More than fifty years earlier, Vannevar Bush [45] had already indicated the main source of this difficulty, and the recent history of serious attempts at introducing a common language for mathematical applications, as well as the older history of false starts, teaches that the problem is even harder than any of them understood.

Not that this was the first or the last time the wish for inter-operability of computer algebra system was brought up. I have a dim recollection of a similar problem being posed

[1] "With a few exceptions (Kant-V2 and Simath or Cayley), almost no interfaces between different [Computer Algebra] systems exist today. I/O, data structures, function calls, graphics etc. are far from being standardized, though users [of CA] would like them to be. This important topic [of standards and interfaces] should be discussed and tackled more thoroughly in the future, on a [national and] international scale." *Translation: Andreas Strotmann. Text in [brackets] added from context to improve readability.*

[2] Originally circulated in 1998, but finally published in *J. Symbolic Computation* in 2000.

during a plenary session at the first CA conference I ever attended, EuroCAL 1985, and the urgency continues to increase rather than decrease as progress is made, but solutions still go through their growing pains.

3.3 Early Beginnings

Early attempts at tackling the problem identified first rather broadly by [45] and later more precisely in [46] and [47] were both motivated and, in hindsight, significantly hampered by the wish to produce a common graphical user interface for Computer Algebra systems.

CaminoReal [48] (1988) is generally cited as the first brave, but ultimately doomed, attempt to build a system that would allow a user to communicate with several CA systems through a common, high-quality user interface. It was an all too typical Xerox PARC project, decades ahead of its time in its vision and thus doomed to fail given the market constraints at the time. It anticipated several aspects of the projects currently most likely to succeed in the application field CaminoReal was aiming at, MathML [37, 15] and OpenMath [1, 10], for example its use of a "code frame" for mathematical symbols the equivalent of which is still under active development at the intersection of the MathML and Unicode [49] standards, and its ability to translate to and from several different CA systems that characterizes OpenMath. However, it was lacking two central aspects of these two more recent attempts: the definition of a common vocabulary that is at the core of OpenMath (CaminoReal professed to provide only a syntactic structure the meaning of which was "in the eye of the beholder"), and the crucial distinction between content and presentation markup that MathML makes.

It was precisely the recognition of this crucial dichotomy of mathematical formulas – namely, the need to specify both the two-dimensional layout and the underlying meaning – that prompted at least one early candidate for an attempt in this direction, the 1994 Text Encoding Initiative [18] specification, to refuse to tackle the problem at all because it was deemed too difficult. Accordingly, SGML markup projects undertaken during those years tended to focus exclusively on the already quite complex questions of specifying the two-dimensional layout of mathematical formulas that appear in scientific publications.[3] Indeed, Bart Wage of Elsevier [50] reported at a 1995 OpenMath workshop that publishers

[3] A review of the field of presentation markup for mathematics or other sciences is beyond the scope of this dissertation.

had despaired at the problem of mixing content and presentation markup in their internal and/or industry-wide publishing standards for mathematical content, and were thus glad to see that the OpenMath project appeared to have a decent shot at solving it.

An attempt that was very similar to the CaminoReal project in many ways, in particular with respect to its focus on a user interface for multiple CA systems, was N. Kajler's 1992 dissertation project CAS/π [51, 52]. However, the focus of that dissertation was on the low-level protocol issues of a "mathematical software bus," and actual communication with external CA systems was done through files containing user-level system-specific commands generated from a CAS/π-internal LISP data structure which primarily served to drive the common user interface. Thus, this project, while interesting as a historical case, is of little actual interest to my current research as it does not appear to have had the concept of a lingua franca, unlike the earlier CaminoReal [48].

3.4 Many Parallel Strands – 1993–1994

It was around this time that Wolfram Research Inc.'s (WRI) MathLink [53] was made available to customers who wanted to link an external application to their Mathematica [54] system, a commercial CA system with (at that time) an exceptional user interface. MathLink was a commercial product about which very little information was made available, and about which it was not very prudent to ask much information, either.[4] About its language component all that was publicized was that it provided a binary representation of a tree structure. While no common vocabulary component was specified, in practice Mathematica symbols were apparently meant to serve this purpose;[5] in other words, other systems were required to conform to the idiosyncrasies of this particular commercial closed system. MathLink came with a specialized protocol designed and optimized for its primary use, namely, to link the Mathematica system with its user interface.

[4]WRI had previously threatened to sue an eminent CA researcher, Richard Fateman, for copyright violations for mimicking Mathematica in a CA research tool that he was making available in an open source mode.

[5]However, given WRI's position as in the previous footnote, anybody trying to connect two non-Mathematica systems using Mathematica vocabulary would open themselves up to the same kind of lawsuit that WRI had threatened before in a highly publicized (and criticized) event. This effectively ruled out the general use of MathLink for this purpose. See [55] for additional reasons not to use MathLink.

Simon Gray, Norbert Kajler, and Paul S. Wang began collaborating in 1993 on the "Multi Protocol" project [56, 57, 58]. Multi Protocol, as the name suggests, focused again primarily on the transport protocol issues and only secondarily on the lingua franca aspect of communication, prompted by its focus on the communication between computer algebra systems and their "dumb" user interfaces. Nevertheless, it has been used to link more intelligent systems to computer algebra systems [59, 60]. Its language component is 32-bit word based, with quite a complex bit field structure per word to avoid wasting too much bandwidth for symbolic expressions. While it does appear to have a notion of common vocabulary, it does not appear to actually define one. Unlike OpenMath eventually, it did not attempt to support multiple linearizations, and lacked appropriate distinctions between language levels, making it unable to make the move from a 32-bit word binary representation to XML when it became expedient to do so a few years later.

Simultaneously and independently, several other early attempts began in 1993. These efforts included ASAP ("A Simple ASCII Protocol") [61], which provided a simple socket interface and an encoding of annotated tree data structures but no attempt at a universal vocabulary, and PoSSo/XDR [62], a SUN XDR-based representation of a class of polynomial representations, both contributions of the European Esprit-funded PoSSo (Polynomial Systems Solvers) project.

My own involvement in this field also began at this time when I started out on a solo project nicknamed CALIF (Computer Algebra Information Interchange Format) [63, 64] which explicitly aimed at devising a lingua franca for CA systems, and in 1993 produced as a first step in that direction a version of the REDUCE [65] Computer Algebra system that could be used for distributed memory parallel processing on heterogeneous workstation clusters, using PVM3 [66] as the underlying transport and process control mechanism, and using a binary machine independent representation of LISP data structures as a prototype lingua franca.

In 1993, Maple Waterloo Software Inc. (Maplesoft), the makers of the MapleV [67] computer algebra system, like WRI before them, were in the process of developing their own technology for separating the user interface from the symbolic computational system kernel, and for letting other systems link to their system. Unlike WRI, however, Maplesoft decided that they needed to have an open standard for this purpose, and Gaston Gonnet,

co-founder of Maplesoft and professor of Computer Science at the Swiss Federal Institute of Technology (ETH) in Zürich, issued an invitation to an inaugural workshop in late 1993 on what he proposed to name the "OpenMath" project. The stated purpose of this project was to define an open ASCII-based standard language and protocol for computer algebra systems to link to each other, and a first draft lingua franca was drawn up and distributed by Stefan Vorkoetter of Maplesoft based on discussions at that first workshop in early 1994 [19].

While the inaugural workshop had had a very limited attendance, news about it spread quickly [6] and all the groups mentioned so far who were actively working in this field [7] were present at the Second OpenMath workshop, which took place as part of the 1994 ISSAC conference, the premier annual international computer algebra conference. Most of these projects decided to merge into the OpenMath group, with the exception of the Multi Protocol project, whose members however did regularly attend the next few workshops, and MathLink, whose authors however made political advances to the OpenMath project to rename their MathLink product to OpenMath – under unacceptable conditions [55].[8]

It was also during this time that the World Wide Web began to rise into prominence around the globe, and given its birth in a physics research lab, it is not surprising that one of the earliest concerns was for adding support for mathematical formulas to HTML. Like the other earlier attempts at defining a common language for mathematics user interfaces, however, Dave Raggett's first attempt in 1994 at defining HTML-Math did not make it [15].

In a different direction, 1994 saw the first international conference dedicated to parallel symbolic computation (PASCO'94 [68]), an important field of applications for a content markup language for mathematical formulas. At this first PASCO conference, [69] presented the design for a distributed heterogeneous symbolic computation system for real constraint system solving, an application that was to become, with my help, the first prototype application for the first OpenMath prototype language and implementations.

Simultaneously, papers began to appear in the automated reasoning community about experiments in joining the complementary powers of theorem provers and computer algebra

[6] I heard of it less than a week after it took place.

[7] That is, with the exception of the authors of CaminoReal – that project no longer existed.

[8] After having been rejected, WRI moved over completely to what eventually became MathML – see the sections below on 1995 and 1996.

systems [70, 71]. Such links were seen as necessary for automated reasoning to handle problems of realistic sizes (otherwise, a theorem prover might spend hours proving that $(a^4)^{10} = (a^8)^5$, say), but on the other hand it was evident from the start that the computer algebra systems' lack of rigor required considerable care when using them in this context. Technically, such experimental links used ad-hoc translators between the syntaxes and semantics of the two participating systems rather than going to the trouble of introducing a system independent lingua franca. Still, the successful collaboration of one major "player" from each respective field – MapleV [67] from computer algebra, HOL [72] from theorem proving – boded well for the future of this particular field of applications.

3.5 Two Bundles and Some Loose Strands – 1995

The year 1995 saw significant progress in the field of content markup for mathematics. Previous parallel research and development efforts were joined into two main "bundles" – OpenMath and HTML-Math – with some "loose strands" of independent efforts such as the Multi Protocol project remaining.

The bundling of several research groups into the OpenMath effort produced an encouraging premiere during the first two months of that year: two different general-purpose computer algebra systems running on different machines cooperated by communicating in a system-independent lingua franca. Based on my previous work [63] with REDUCE, and a prototype OpenMath implementation for MapleV by Stefan Vorkoetter of MapleSoft, we were able to demo at the third OpenMath workshop a system with a REDUCE [65] process as the "master" and a MapleV process as the "slave" communicating via PVM3 [66] in an extension of Stefan Vorkoetter's prototype draft OpenMath language [19] (see Chapter 5 for details). At the same workshop we presented a draft of the OpenMath Objectives paper [6, 26], the final version of which was presented later that year as a poster at the ISSAC'95 conference [7].[9] This paper contained an analysis of the requirements for an OpenMath language, including several use-case scenarios, and a proposed architectural overview of such a language based on parallels to the linguistics of human languages.

[9]This paper appeared in 1998 as [1].

Based on that first OpenMath MapleV–REDUCE connection, Ph. Marti and I implemented a first proof-of-concept real-life application using OpenMath as a lingua franca in a cooperative distributed system of symbolic computation systems, adding three more heterogeneous systems into a complex framework of cooperating distributed agents [73, 74] that summer.

At the end of 1995, the OpenMath group began a formal process of writing an actual OpenMath definition [10] based on the recommendations of the Objectives paper [7].

On the HTML-Math front, a "panel discussion on mathematical markup was held at the WWW Conference in Darmstadt in April 1995. In November 1995, representatives from Wolfram Research presented a proposal for doing mathematics in HTML to the W3C team" [15]. In contrast to the pure content-markup approach of OpenMath, the Wolfram Proposal (as it became known) represented essentially a pure presentation-markup ansatz,[11] with a special no-angle-brackets syntax for formulas to be embedded in HTML. Though this particular proposal would undergo radical changes over the year, it resulted in the successful formation of the second main bundle of the parallel research and development strands in mathematical markup, the MathML effort.

The Multi Protocol (MP) project continued as an independent effort, but being tied to a binary 32-bit-word-unit model and lacking an independent language design, it was not able to contribute much to either of the two main "bundles" of OpenMath and MathML, both of which were aimed specifically at the Web. Nevertheless, the MP authors continued to present the experience they gained at the OpenMath workshops during this year.

3.6 Regrouping – 1996

Thus, at the end of 1995, OpenMath was well advanced, HTML-Math was just being revived after its 1994 debacle, and MP continued to follow its own course. The year 1996, however, saw some radical changes in this scene.

First of all, for good political reasons, the focus shifted almost completely to the newly formed HTML-Math group, especially among the commercial interests in the OpenMath

[10] Robert Sutor, editor, and John Abbott, Andreas Strotmann, André van Leeuwen, Stefan Vorkoetter, James H. Davenport, authors. [75]

[11] WRI researchers have consistently maintained that presentation markup is enough, and that their Mathematica system at least is able to deduce content from presentation markup.

group. To counter the commercial influence of WRI and what was deemed a dangerous over-reliance on presentation markup, several of the larger OpenMath member groups – especially those from IBM, Maplesoft, and INRIA[12] – became members of, and contributors to, the W3C's HTML-Math working group when it was formally chartered as MathML in the second half of 1996. It is due largely to these members of both OpenMath and MathML[13] that MathML now contains a viable content markup component in addition to the strong presentation markup part based on the original Wolfram proposal.

Thus, the two main bundles of development in this field actually merged together, and mutual compatibility has continued to be a major factor in the development of either of the two consolidated efforts.

Second, the writing of an OpenMath definition came to an abrupt halt when it had barely begun. Several factors contributed to this sudden change, but two of these were crucial. Most important, the editor of the document and a large portion of the OpenMath membership shifted their focus to the W3C HTML-Math working group where they submitted within just a few months what came to be known as the INRIA Proposal, which would eventually evolve into MathML-Content. In addition, among the main authors of the OpenMath definition, personal tragedies forced me to abandon work on OpenMath completely at that time, and the remaining authors were not able to take up the slack left by Robert Sutor and myself.

In an heroic effort to save the project, Gaston Gonnet, in collaboration with A. Diaz (the latter eventually became co-chair of the MathML working group), did put a draft OpenMath definition on the Web[14], but this version fell short of the goals originally set by the originally designated authors of the definition.

Politically, as promising as things had looked a year earlier, at the end of 1996 things looked grim indeed for OpenMath.

Third, the Multi-Protocol project was pushing forward quite strongly in 1996, with journal and conference publications on the protocol [57] and its applications [59, 76] following a first public release of the software.

[12]INRIA had recently taken over from CERN as the European site of the World Wide Web Consortium (W3C).

[13]At the time of this writing, one third of the total MathML working group membership also belongs to the OpenMath group, as did all but two members of the original MathML Content sub-committee.

[14]This set of documents has been taken off the Web completely now, and can unfortunately no longer be re-constructed.

A fourth contender called MathBus [77] (R. Zippel, Simlab Project, Computer Science Dept., Cornell University) made a brief appearance on the scene at this time, in the context of the Weyl CA system. The language design they propose has a close affinity to the design outlined a year earlier in our OpenMath Objectives [7], covering essentially the data structure and semantic structure ("expression") levels of that document. Indeed, OpenMath was MathBus's designated textual form.

Outside of this core area of the content markup for mathematics field, distributed cooperative symbolic computation application prototypes that cried out, both explicitly and implicitly, for a lingua franca and common protocol were becoming more numerous, and the implications and complications of such cooperative schemes were being scrutinized more closely.

With respect to linking theorem provers and computer algebra systems, work continued on combining HOL and MapleV [78], and another group worked on combining Isabelle and MapleV [79, 80]. The multi-solver constraint logic programming system of [69] that had served as a first OpenMath prototype application reached fruition in 1996 with the publication of Ph. Marti's dissertation [74]. Cooperation between symbolic logic and symbolic algebra systems was thus beginning to emerge as a core application area for a mathematical content lingua franca, and as an important research area in and of itself. The Calculemus[15] series of workshops began in this year, though it would take another 2 – 3 years for the field to mature and for these workshops to become fully refereed international conferences with formally published proceedings.

Nevertheless, several papers appearing at major conferences at this time discuss the general design and framework of such cooperative computing systems. [81] describe a framework that amounts to a Problem Solving Environment for automated reasoning dubbed "Open Mechanized Reasoning System" into which a wide variety of systems might be plugged. [82] explore the design of an "object oriented" parallel reasoning system. [83] argue for hybrid knowledge representation schemes in the context of cooperative distributed agents that have a wide range of different perspectives on the problem they collaborate on.

[15] www.calculemus.net

3.7 Consolidation – 1997

In 1997, the OpenMath and MathML projects merged and then split again. They merged in the sense that there was now a large overlap of people in both groups, and they split in the sense that the MathML group internally split into two parts along conceptual lines. The MathML recommendations were divided into what the project dubbed "presentation markup" on the one hand and "content markup" on the other. Almost all MathML Working Group members working on the latter part of MathML were at the same time members of the OpenMath consortium.

Politically, this must have been a strange affair. "Content Markup" was certainly the less "clouty" branch of the two, with much less ambition, too. Indeed, the most forceful proponents of MathML-Presentation, WRI, had long argued that MathML-Content was unnecessary. However, major developments taking place during that time in global World Wide Web politics added strong support to the proponents of a Content encoding branch from an unexpected quarter.

The World Wide Web Consortium (W3C) under Tim Berners-Lee, the "father of the Web," had gained considerable political weight by this time, and the political situation of the Web in society was given particular weight within the new-found role of the Consortium. In particular, treatment of Web accessibility issues became of paramount importance within the quickly exploding web of inter-related W3C working groups, and it was easily shown that for mathematical formulas in particular, MathML-Content would provide a much better basis than MathML-Presentation for improving the accessibility of Web documents with mathematical content, be it through audio rendering or through multi-lingual support [84].

Another factor of major influence in the grand scheme of things at this point was the introduction of XML, the development of which was taking place simultaneously with that of MathML and was heavily influenced by MathML as a prototypical – or rather, worst-case – XML application.

Politically, MathML had no choice but to become an XML application; native support for special-purpose mathematical markup, as was frequently advocated by opponents of the admittedly verbose XML markup, was not going to be forth-coming from any off-the-shelf Web browser, and an XML encoding would ensure that a MathML formula could at least

be read syntactically by all future Web browsers. XML stylesheets for MathML might then allow its reasonable-quality rendering in any browser that might not have any native MathML support.[16]

The closely inter-twined development of XML and MathML meant that the MathML 1.0 Recommendation [14] was published just a couple of months later than the XML 1.0 Recommendation [37].

During this time, although the focus was very much on MathML, the OpenMath side of the joint MathML–OpenMath content markup project was gaining strength again. For one thing, the scope of MathML-Content was officially limited to "K-12 and the first two years of college in the US" [14] in its mandate, while OpenMath's aim was very much towards more advanced mathematics, too. Thus, from the start, OpenMath was being designated in MathML-Content as a mechanism for eventually extending the scope of MathML.

A second important step for OpenMath was that it finally managed to become a well-funded project, with considerable EU support for a three-year "multimedia" project that ended in late 2000. Among the people hired for this project was, in particular, David Carlisle, an editor of the LaTeX2e system, who would eventually co-edit both the MathML and the OpenMath standards in addition to his already prominent position in the LaTeX community.

From a more technical perspective, since OpenMath had opted to support an SGML encoding from the start [6, 7], the move to an XML encoding was a natural step.[17] It was also clear that OpenMath would not be able to have the same XML syntax as MathML, due to the restrictions imposed by XML that forced MathML to restrict itself to a closed set of mathematical concepts as opposed to the open-ended library of such concepts that OpenMath was aiming to support. Thus, development of both standards, while heavily interrelated, could and would proceed independently to a considerable extent.

[16] This is a paraphrase of a position that has been taken by many different members of the MathML group at many different times.

[17] However, this step did mean that G. Gonnet's earlier SGML-based draft version of OpenMath would need to be revisited since it utilized SGML features that were no longer available in the XML simplification of SGML.

3.8 Cleaning Up and Shaking Down

In early 1998, the World Wide Web Consortium published the Extensible Markup Language (XML 1.0) [37] and Mathematical Markup Language (MathML 1.0) [14] Recommendations in quick succession. Later that year we found that there were still some fundamental problems with both the MathML content markup and with the then-current OpenMath draft language designs [2, 21], but at the November 1998 OpenMath workshop at Florida State University, the Esprit Project OpenMath presented for comment and approval by the OpenMath community its new draft OpenMath language which, based on a fundamental concept that I had introduced into the discussion a few years earlier, finally addressed at least one of the fundamental problems that had been plaguing essentially all earlier designs of content markup languages: it achieved compositionality [3]. The OpenMath standard was finally published in early 1999 [10].

Since then, the OpenMath core language specification has been stable, but the design and specification of its basic vocabulary of mathematical concepts has undergone some fairly radical changes. During the year 2000, the review process for this part of OpenMath still uncovered scores of problems.[18] In the end, the core Content Dictionaries were "formally approved" in a version 1.0 while some of these had still not been addressed properly.

At the point of this writing, some minor and major errata in the OpenMath 1.0 standard are being fixed, and proposals are being made to add features that have been missing from it, such as the view of OpenMath Objects as a directed acyclic graph structure rather than as a tree that we had proposed in the OpenMath Objectives in 1995. Content Dictionaries are also being developed to cover a wider range of mathematical concepts.

In a similar vein, the basic MathML-Content design has been quite stable since its 1998 publication, but the concrete vocabulary of mathematical concepts in that standard had to be reworked considerably for version 2, incorporating among others many of my suggestions for additions as well as deprecating some "features" that I had discovered violated the compositionality principle in a particularly blatant manner. Along with many other updates, this led to the publication in 2001 of MathML 2.0 as a W3C Recommendation.

[18] I was the main "external" reviewer, see `www.cs.fsu.edu/~strotman/projects/OpenMath/CD-reviews/cd-review.html`.

Both projects had to pay the price of needing to stay in synch with the ongoing XML effort. Several clean-up tasks needed to await the resolution of issues in other XML subcommittees. XLink [39], or the XML Linking language component, is a case in point, where the exceedingly important aspect of substructure sharing that was identified as a critical component of OpenMath from the start [7], was left out of both standards in order to wait for a common XLink resolution.[19] Similarly, the XML Schemas [36] and XML Namespaces [41] projects were still dealing with issues in extensibility and embeddability of XML documents long after these problems had been identified as crucial for the success of both the MathML and the OpenMath project. It is therefore not unlikely that future updates to both of these will be needed as related XML standardization efforts reach fruition in the coming years.

However, both projects also profited from other W3C XML sub-committees. In particular, the XML Stylesheets working group [38] produced valuable tools for bridging the gap between MathML Content and MathML Presentation Markup, or if necessary between the specialized mathematical and the general-purpose layout markup available in a browser. Several papers have explored style sheets for the purpose of translating from content to presentation markup, and for the purpose of translating between OpenMath and MathML (e.g. [85] (1998), [86] (1999), and [87] (2000)).

A style-sheet based conversion of content markup to presentation markup turns out to be very complex, but recent developments are likely to reduce the extent to which the success of at least MathML will depend on such style sheets: a member of the Microsoft browser development team has joined the MathML working group, indicating a commercial interest in teaching their browser to natively support at least MathML Presentation markup, and in 2001 the Mozilla open-source version of the Netscape browser had already been shown to support native MathML-Presentation, and has been boasting fairly complete handling of MathML 2.0, both Presentation and Content Markup, since 2002.

[19] MathML 2.0 now includes references to XLink as a mechanism for synchronizing MathML-Presentation and MathML-Content versions of the same formula in a single MathML document. M. Kohlhase is now reviving the push for including structure sharing mechanisms in OpenMath.

3.9 Onwards

At the point of this writing, MathML is close to reaching the goal of mainstreaming the representation of mathematical formulas into the World Wide Web, with support available from major browser vendors as well as the computer algebra systems world. OpenMath, by contrast, is still undergoing some cleaning up in its core language component, although the proposals are basically backward compatible. Its Content Dictionaries, of course, are an open-ended task by design, unlike MathML's Content markup, and new mathematical concepts are being added constantly. However, both projects are now remarkably stable, and after a long story with many up-swings and down-turns, they are today together the most promising candidates for long-term standards for mathematical document delivery on the Web.

With the recent publication of draft standards for XML and HTTP-based collaborative computing environments such as SOAP [88], they are also becoming very promising candidates for the foundations of Web-technology-based problem solving environments for areas with a significant amount of mathematical content [89]. Indeed, already some proposals have been published that provide plans for extending the combination of MathML and OpenMath to applications beyond their current scope. One particularly active group in this respect comes from the Theorem Proving community again, where Michael Kohlhase and his European collaborators have been working on a project they call OMDoc [90] and on the use of OpenMath in an environment of heterogeneous automated reasoning systems [91]. In Japan, the OpenXM project [92] is working on building an open-source problem solving environment for mathematical problems that provides for OpenMath processes to be hooked into the system. Trying to extend the applicability of the OpenMath/MathML approach to sciences beyond mathematics, [89, 28] have been collaborating on a project to build a problem solving environment for coupled climate modeling.

CHAPTER 4

RELATED TOPICS

Unfortunately, there is a huge amount of literature in the wider research area of our dissertation even if only closely related topics are considered, making full coverage of the literature impossible in the framework of this chapter. On the one hand, we therefore need to make rather arbitrary delineations on the literature we cover, but on the other hand a full survey of the field should be done regardless.

In the light of this dilemma, I propose in this chapter to survey related topics without going into too much detail as to the concrete literature covering them.

4.1 Applications and Environments

A wide range of applications would benefit from a well-designed content markup or knowledge communication language. Many such applications have been proposed, and many have been tried.

4.1.1 Collaborating Symbolic Computation Systems

The core application area for such languages is that of an *interlingua* in heterogeneous distributed and parallel symbolic computation efforts. Examples of these are a multi-solver constraint-logic programming system [69, 73, 74] or the MathWeb system of cooperating proof systems [91]. In Artificial Intelligence, the field of intelligent cooperating agents may be viewed as an example of distributed symbolic computation, and both the older efforts of the DARPA Knowledge Sharing Effort, including the Knowledge Interchange Format (KIF [93, 94]) and the Knowledge Query and Manipulation Language (KQML [95, 96]), and the very recent DARPA Agent Markup Language (DAML [97]) are noteworthy communication languages that have been used in this area.

4.1.2 Symbolic-Numeric Interfaces

In a more general scheme, interfacing between systems in a more heterogeneous and more loosely coupled environment through a common language is an important application area for content markup languages. In addition to symbolic computation systems, numerical and statistical tools, graphical user interfaces, visualization tools, audio and visual rendering engines -have all been considered to some degree or other in such an environment. The Multi Protocol project [56, 57, 58] has been particularly interested in such interfacing, especially with GUIs and visualization tools. Mike Dewar's 1990 IRENA (Interface between REDUCE and the NAG library) [98] was an early example of an interface between symbolic and numeric systems. WRI's MathLink and Maplesoft's interest in OpenMath both evolved from a need to split the core symbolic computation engine of the computer algebra system and its user front-end.

4.1.3 User Input and Output

[84] explores the use of audio rendering and argues that content languages are better suited as input material to such a process than are presentation markup languages. At the user input end of the pipeline are those exploring the possibilities of automatically deriving content markup from scanned images [99], from online analysis of handwriting [100], from TEX or LATEX source files, or from other more or less raw data sources – each a highly specialized technology that needs a common language to translate into in order to succeed in finding clients to deliver to.

4.1.4 Symbolic Computation Web Services

Many papers in the References discuss ways and means of providing services on the World Wide Web that would require a common language in which to speak to their clients. G. Gonnet presented a "Web Pearls" approach to offering access to computer algebra as a web service at a 1998 OpenMath Workshop, for example [101]. In the Multi Protocol group, O. Bachmann describes how he put a highly specialized and optimized server for Chains of Recurrences onto the web via the MP protocol, and P. Wang [102, 103] discusses ideas of going beyond MP to an "IAMC design" (Internet accessible mathematical computing). [104, 105]

report on a project that makes another newly discovered computer algebra technique, the fractal Gröbner walk, available on the web. [89] discuss the use of several web- and XML-based techniques for building a web-centered problem solving environment.

4.1.5 Interactive Textbooks

In interactive textbooks, both on CD-ROM and for distance learning via the web, a common representation language for the content being taught is of fundamental importance. Interactive components require quite sophisticated algorithms tailored to the special requirements of a learning environment, as a recent workshop at Florida State University showed [106]. Groups that are investigating this part of the application spectrum for content markup languages include, from a computer algebra background, Arjeh Cohen et al., whose interactive introductory college algebra book [107] was recently published by Springer, and from an AI background, an Interactive Textbook project at Saarbruecken University [108].

4.1.6 Communication Architectures

A common theme in the literature that is closely related to our research topic concerns other aspects than language when considering the interactions between computing systems. All kinds of information broker architectures have been proposed, and it is clear that content-based communication languages do make for special problems in this regard. The mathematical software bus concept appears to crop up again and again in this context [52, 105, 109], as do client-server architectures [19, 102, 92]. Another interesting aspect of this theme is that of object models for languages in the XML group, including MathML, and the question of how content-aware object models might differ from purely syntax-based ones.

4.1.7 The "Semantic Web"

Another exciting field that is currently under investigation without having reached critical mass yet is that of software agents, in particular mobile software agents on the web. According to Tim Berners-Lee[110] "machine-understandable" web content, i.e. web content marked-up with algorithmically determinable semantics, is a necessary substrate for

the dream of web software agents to become reality; upcoming conferences on the "Semantic Web" will be dealing with related issues. The prototypical language in this field is the Resource Description Format (RDF) [111].

4.1.8 Mobile Communication

Perhaps surprisingly, it turns out that extending the concepts of the World Wide Web to mobile devices as user interfaces is an important application area for content markup languages. Due to their low bandwidth to the user's attention, as well as to their multi-modal nature, adapting web content to such devices in addition to the regular web is much easier to do if the adaptation process is based on a semantic structure representing the actual information that is to be conveyed to the user. Conceptual modeling (i.e. cognitive science) plays a crucial role in this area, and content markup or knowledge communication languages are crucial for system and device independent representations of users' conceptual models.[1]

4.2 Existing Markup Languages

An important aspect of content markup languages is that, alone, they are of very little use: without ways of presenting content in a consistent and user-friendly manner, be it through audio- or through visual channels, that is without corresponding presentation markup languages, their use is confined to inter-system communication. Similarly, a pure content markup language that does not pay sufficient respect to the underlying general-purpose or special-purpose data structures used for scientific data formats loses some of its appeal in applications that are not purely symbolic in nature. On the other hand, separating these three aspects of otherwise tightly coupled problems is a crucial aspect in the design of systems that contain a content markup component, just as interweaving these components into a cohesive whole becomes a major issue.

MathML 2 [15] is to be particularly commended for making, on the one hand, the crucial distinction between content and presentation markup very explicit – indeed, enforcing it –, and for treating extensively, on the other hand, the questions of inter-weaving these two aspects (notion and notation) of the subject matter they deal with. This problem had

[1] Based on a seminar presentation at CSIT the author and title of which I cannot reconstruct. It described the author's work on a mobile geographic information system.

previously been discussed by S. Dooley [112] in the context of experience gained implementing MathML for IBM's TechExplorer [113].

A similar analysis of the problem of mixing content markup and scientific data formats has, to my knowledge, not been dealt with in such a compelling manner yet, although our OpenMath Objectives [7, 1] provided a promising ansatz in this direction, and the Multi Protocol project [56, 57, 58] made interesting contributions to this aspect of the field.

A closely related experiment in mixing and matching languages for a large number of different levels, purposes, and applications is taking place in the form of the whole family of XML standards, starting with the Unicode [49] character-set underlying XML, through XML syntax [14] and DTDs, via XML Links [39], Pointers [40], and other fundamental data structures (XML Schema [36]), to presentation languages like XHTML [34]and SMIL [114], scientific data formats like CML [115] and BioML [116], and content markup languages like MathML and OpenMath.

4.2.1 Symbolic Computing Languages in Artificial Intelligence and Computer Algebra

In this section we deal with the perhaps most obvious close cousins of content markup and knowledge communication languages: user and system level languages and formats used in symbolic computation systems.

In the wide field of AI, knowledge representation system languages come to mind immediately. The DARPA Knowledge Sharing Effort was a major research project that undertook to understand and implement mechanisms for sharing knowledge among distributed intelligent agents. This project, which started a couple of years before the OpenMath and MathML projects, has produced drafts for a variety of tools for knowledge communication of AI agents.

Of these, the Knowledge Interchange Format (KIF, [93, 94]) is certainly the most interesting subproject of the Knowledge Sharing Effort in the context of this survey. KIF has been a standard in the making[2] for distributed knowledge based systems for a few years longer than OpenMath, and it would be an intriguing exercise to study the history of this project in the way that I went over the joint history of OpenMath and MathML in Chapter 2.

[2] [94] (1998) is the document for a draft proposed American National Standard, while KIF version 3.0 [93] dates back to 1992.

There is evidence that KIF has gone through similar stages as these two have; in particular, the differences between KIF 3.0 and dpANS KIF are enormous, and telling. In Chapter 7 we analyze in detail some strong limitations of KIF in either version when considered as a general purpose content markup or knowledge communication language.

A second result of the Knowledge Sharing Effort was a draft agent communication language, KQML (for Knowledge Query and Manipulation Language). A simple, though not fully precise way of characterizing the difference between KIF and KQML is that KIF is about semantics, but KQML about pragmatics of inter-agent communication.

Somewhat in between these two is another subproject of the Knowledge Sharing Effort, the Ontolingua project, which aimed to define shared knowledge representations and a common vocabulary for common concepts in Knowledge Engineering. This is quite similar to the concept of Content Dictionaries in OpenMath, but with a much more formalized semantics than OpenMath. However, ontologies for knowledge representation are orders of magnitude more detailed in the knowledge structures they represent than a knowledge communication language should be; typically, only what is known in that field as "top-level" ontologies should be used when communicating with other intelligent agents, and OpenMath Content Dictionaries tend to be exactly that (minus the formal semantics) in the case of mathematics.

A special mention should be made at this point of the Language Engineering branch of Artificial Intelligence. One of its core components is the design and implementation of natural language user interfaces to databases or other tools, both as input and as output channels. A particularly interesting aspect of this branch of AI in the context of this survey is a set of initiatives for "natural language markup" – markup conventions for straight-line text that range from full grammatical analyses of the marked-up text corpora, usually of interest mainly to linguists, and thus perhaps an equivalent of MathML or OpenMath for that field, to mark-up of semantic and pragmatic speech patterns in such corpora. A particularly interesting project here is the Universal Networking Language (UNL) project led by the United Nations University [117]. The quite ambitious goal of this project is to provide a formal universal language that can be used to encode the sentence and word structure of text in any human language, with fully disambiguated word meanings and in such a way

that a translated version of the text can be reproduced in any other language supported by the system.

In the field of Computer Algebra, any computer algebra system's user-level language qualifies as a content language of its own right – MathLink, in a sense, relies on Mathematica for this purpose, for example –, and CA user and system languages are thus of close interest to our research. Indeed, in [2] we went into some detail analyzing the suitability of these languages as more general-purpose content markup languages, and found that all of them have some fatal flaws in their designs that a general-purpose knowledge communication language must avoid.[3]

Needless to say, any of the communication languages and systems built for linking mathematics-based tools – MathLink, Multi Protocol, OpenMath, MathML, ASAP, PoSSo/XDR, or Camino Real – are very closely related to my research and need to be studied carefully for their pros and cons.

At this point it may be useful to point out the peculiar way in which a distinction between the CA and the AI view of mathematical formulas manifests itself. It turns out that CA is most interested in equivalence-preserving transformations of mathematical "terms", while AI reasoning systems tend towards a pure logic view of things where the only part of interest in a mathematical formula is its "propositions." In this sense theorem provers and CA systems are orthogonal to each other, each bad at what the other is good at. This fundamental difference also makes for differences in the semantics of communicated information, a problem that every single paper about attempts at linking CA into theorem proving or logic programming is bound to bring up. This problem is commonly summed up by saying that the results produced by a CA system can not be trusted and have to be checked before integration into a knowledge base.

4.2.2 (Semi-) Numeric Computations: Scientific Data Formats

Another more distantly related cousin of our field of research can be found in the field of Scientific Data Formats. Like knowledge communication languages, the intended use of

[3]See also [3] where we expand on this argument and show that the new OpenMath design is the first language that qualifies in this regard, not least due to my own efforts in guiding its development in this direction and insisting that the fatal flaws in the design of the big CA system user languages not be carried over into the OpenMath effort.

these data formats is the exchange of information between different computational systems with overlapping capabilities. This area has been well studied and quite successful during the last decade or two.

Examples of this field include the following. In computational physics, netCDF [118] and its cousins are well-established; in chemistry, CML [115] is a promising candidate; in bio-chemistry, BioML [116] is an attempt to establish a scientific data format for gene sequencing and related data. OpenGIS [119] is beginning to play the role of a standard data format in the geosciences. In biology, though I am not aware of a scientific data format proposal, there appear to be projects underway for global taxonomic and ecologic databases that could benefit from such a format.

It is quite instructive to consider why this is a distinct research area from that of content markup and knowledge communication, even though three of the four examples mentioned above (CML, BioML, and OpenGIS) are bona-fide markup languages (i.e. XML applications). The distinction is made here between data or information on the one hand and content or knowledge on the other. In general, scientific data formats tend to be only lightly structured and quite rigid, closed data formats that are meant to mirror a database design with many implicit and tacit assumptions, e.g. about data types, built in. By contrast, the structure in a content markup tends to be extremely flexible, completely extensible, and with much of the knowledge explicitly stated as markup.

Nevertheless, both fields can certainly profit from each others' work. Data-mining in (semi-) numeric collections of data leading to highly structured symbolic representations of knowledge or conjectures about the scientific data found, is one example of an application, especially if the knowledge thus derived could be added to the database in a common knowledge format.

4.2.3 Textual Information: Meta-Data and Text Encoding

When publications in our research field discuss related work, they tend to begin, depending on their authors' backgrounds, either with TEX and LATEX, or with SGML- or XML-based standards like HTML or the Text Encoding Initiative's recommendations [18] for marking up a text with its textual structure.

As discussed earlier, the focus on providing the user with the visual form of mathematical information doomed several early attempts in this field, but a lack of treatment for this aspect of mathematics has been nearly as catastrophic for other projects since the ability to examine results in an intuitive format is a fundamental requirement for any system for doing mathematics, indeed, for any system for doing any of the natural sciences.

On the other hand, systems that were purely meant for typesetting or desktop publishing were, on their own terms, most successful. LaTeX, for example, is to this day one of the most popular means of publishing papers in the natural sciences, and no-one needs a reminder of the immense success of HTML. They tend to have everything that is needed to quickly author and pleasingly render to the discerning eye any formula in mathematics and the natural sciences, but – they make it extremely hard to re-use the knowledge embedded in those formulas in any other system. Even if in principle those systems are able to handle the content of those formulas, correctly extracting that content from the presentation markup is generally considered to be a problem on a par with natural language understanding [45, 21], that is, way beyond current means. This does not mean that there are no projects attempting exactly this, but then the whole enterprise of Computer Algebra and Artificial Intelligence is one that is aiming to do the impossible,[4] and do it well.

Nevertheless, the requirement for content markup to be useful both for World Wide Web browsers to display and for easy re-use of the knowledge thus encoded, say, via copy-and-paste to a computer algebra system or proof checker, precluded relying on artificial intelligence techniques for disambiguation of content, and thus precluded extending the use of existing markup languages like LaTeX to this new application area except via definition of a set of "semantic macros" that mirror a bona-fide content markup language.[5] For this reason, the area of text markup, while of interest when it comes to the need to interface with it, is not of central concern to the research reported here.

[4]The constant problem, namely the question whether or not two symbolic constants are identical, obviously a subproblem of most any computer algebra algorithm, is known to be undecidable in principle, and one of the most important methods in computer algebra, the Groebner basis method, has a super-exponential worst-case space complexity.

[5]B. Miller, D. Carlisle, personal communication.

The matter is different with another interesting "classic" of text document processing, the field of meta-data. At first glance, this field is tightly coupled to knowledge communication, as it ostensibly deals with encoding knowledge about the documents it is attached to.

Meta-data is an old classic in library cataloging; MARC [120] has been a document meta-data standard for decades. More recently, the Dublin Core project [121] defined a simplified cataloging format for web documents.

Interestingly, mathematicians played a major role in developing this new standard, and it is quite instructive to consider what prompted them to collaborate in the Dublin Core meta-data format standardization process.[6] Their basic idea was to provide a clean user interface to a nation- and world-wide collection of paper preprints [122], all of which were available in LaTeX form. The main problem that they needed to solve was that there appeared to be no reliable method to search the content of these files and get meaningful results, a problem that was obviously closely related to the one mentioned above that precludes TeX's use as a content markup language. To circumvent this problem, they chose to provide methods for adding a particular kind of meta-data to each preprint's entry: a paper's classification according to a mathematics classification scheme (MSC). As a consequence, this group was one of the first world wide with working Dublin Core data bases and search engines.

It is thus in the semantic aspects of meta-data that our research interests meet. Indeed, the interests of the meta-data community in semantics is growing quickly, and goes beyond classification. One of the most prominent proponents of this field is the "father" of the web himself, Tim Berners-Lee, who has published a series of papers on his concept of a "semantic web" [123]. Web standards such as the Resource Description Format (RDF, [111]) and the Dublin Core play a major role, as do first applications of these such as the W3C's Privacy Protection Platform [124]. There are only two main differences, it seems, between our effort and this one: first, we are interested in *including* semantic information in a document, bringing the document itself to life, while the meta-data efforts try to *add* semantics to a document from the outside, as it were; and second, we are coming from a formalization of

[6]The following discussion is based on personal communication with members of a consortium of Institutes of Mathematics in Germany, among others Kallies and Schwänzl.

highly technical and precise sciences, while the Semantic Web idea has focused primarily on ways of enabling social contracts, as it were.

It will be interesting to see to what extent these two approaches will be able to grow together eventually. One idea of how this might be accomplished I discussed with Robert Cailliau, known as the co-father of the web, at an inaugural workshop for the German section of the W3C in 1997. In this scenario, the intermediate step between the meta-data approach and the content markup approach could be a "formal abstract", that is a summary of the document in the form of a "mathematical formula" encoding the main points of that document. At any rate, an integration of MathML and OpenMath into the RDF framework would be a very worthwhile undertaking.

Thus, the fields of text structure markup and meta data are fields that are distinct from content markup, but there is great potential for mutual profit. Content markup would clearly add value to these two fields, and content markup without these two as a context, while useful, is clearly lacking important aspects.

4.2.4 Application-Specific Data Formats

Another large and diverse field of study that is related to our research interest, and in many cases interwoven with entries in previous sections, is that of application-specific data formats. Examples in this area would include the Graphical Kernel System (GKS) data representation language for computer graphics, any of a wide variety of image, audio and video formats (e.g. GIF, JPEG, MIDI, AIFF, MPEG), virtual reality descriptor languages such as VRML, and similar languages that describe fairly directly multi-media content as it is to be presented to a human user. This latter aspect distinguishes these languages from scientific data formats that are primarily meant to be read and written by (semi-) numerical software systems. A full review of these data formats is outside the scope of this dissertation.

4.2.5 General-Purpose Data Formats

The most prominent members of the group of general purpose system-independent data formats are Sun's External Data Representation (XDR) and the ISO standard Abstract Syntax Notation ASN.1 with its Basic Encoding Rules. Both are primarily used for the

definition of networking protocols; indeed, XDR's origin is as the data representation for Sun's Remote Procedure Calls (RPC). Both are binary data formats, unlike the text-based markup formats like XML. Both share the property that an application's bit streams are impossible to parse correctly even syntactically without precise knowledge of the data structures exchanged using these mechanisms, again quite unlike XML streams. The latter is perhaps the main problem for the use of ASN.1 as a basis of a binary encoding for content languages, as I had originally proposed in [64]; otherwise, ASN.1 offers data representation support comparable to XML (standard identifiers, sequences, record structures, date/time), with additional support for numerical types (arbitrary size integers, floating point numbers) that XML lacks in and of itself. XML Schemas and SOAP provide two XML-based general-purpose data formats that are comparable in their extent to ASN.1.

4.2.6 The Extensible Markup Language

The W3C's Extensible Markup Language (XML) group of languages deserves a special mention. For one, it is the syntactic basis of a significant number of languages that are of interest here (MathML and OpenMath are prominent examples). Second, a host of languages of interest here are in the process of moving towards an XML syntax (OpenGIS and HTML are examples). Third, XML is interesting in its own rights for the syntactic infrastructure it provides, and for the way this infrastructure is spread over many XML building blocks – DTDs, schemas, name-spaces, links, pointers, and so on – and how this complex structure of inter-related standards is still constantly and rapidly evolving.

It is when considering the often bewildering complexity of the ongoing XML effort that it becomes glaringly obvious why the OpenMath effort took such a long time to mature: the syntactic framework was simply not available that would have been required as a starting point for OpenMath to grow from.

4.3 Theoretical Background

There is a wide range of literature on formal models of communication. In addition to Claude Shannon's famous foundational theories of information based on concepts of entropy – that is, models of the hardware characteristics of the communication channels – there is an

easy distinction made between syntactic, semantic, and pragmatic layers of communication, and different models may be classified with respect to the levels they treat. Needless to say, in this dissertation we are mostly interested in the semantic layer and beyond, but it turns out that it is necessary to include syntactic issues at least to the extent that they interact with the semantics layer. Within the semantics layer, another distinction that needs to be made is between structural semantics and lexical semantics.

4.3.1 Language, Logic, Semiotics

Mathematical models of the syntax and semantics of formal languages are mostly treated in the Logic and Decidability and in the Model Theory branches of mathematics. Other branches of mathematics that are of importance here include Category Theory, Higher Order Logics, Lambda Calculus, and (especially for the pragmatics layer) Modal Logics. In other words, the branches of mathematics that are relevant here are intimately connected with the foundations of that field.

The mathematical theory of formal languages and parsing is only of peripheral interest here, however, since the syntactic component of a content markup language tends to be kept as trivial as possible. Even the formal semantics of programming languages are of limited relevance only, since content markup is clearly primarily concerned with static semantics, and oftentimes with informal rather than formal semantics as well.

A common notion in mathematical language models is that of Interpretations and Representations as mutually inverse mappings between the syntactic and semantic incarnations of an expression. Rewrite and production rules are a common model frequently used in both the syntactic and the semantic analysis of such expressions.

Such formal models offer a great deal of help when it comes to fundamental properties of languages and their meaning; however, only fairly recently have research efforts begun that study semantics at a sufficiently fine level of granularity to offer support in understanding the issues involved in languages with realistic ranges of applicability. Most mathematical models, unfortunately, tend to simplify their subject matter to a point that their practical relevance is all but lost. The study of the mathematics of formal languages, for example, which was pioneered by Noam Chomsky as an attempt to understand the syntax of natural language with some remarkable mathematical results, ignores most of the fundamental

aspects required by a realistic theory of natural language syntax like Noam Chomsky's much more recent theory of Government and Binding. Similarly, research into the formal semantics of natural language has led to very recent major advances in mathematical logic [25].

4.3.2 Linguistics and Cognitive Science

Mostly for this reason, we need to investigate the literature on the linguistics of natural language, and on cognitive science, to find a reasonably detailed theory of semantics for knowledge communication. The basic premise here is that the only known high-quality solution[7] to the knowledge communication problem in a heterogeneous environment is our human system of communication, i.e. the human language capacity, and that while formal models may help understand some of its properties better, these models need to be informed by the actual structure of communication between people as it is.

Communication models are studied in a rather multi-disciplinary array of fields. Linguistics, be it of the formal, computational, or "hunter-and-gatherer" variety,[8] is one such field. Cognitive science, philosophy of language, semiotics, artificial intelligence, and advanced mathematical logic are just a few additional research areas studying models of communication.

Fortunately, linguistics has matured enough during the last fifty years or so for some fundamental results to be mostly agreed upon, and recent years have luckily seen the publication of very accessible (to non-linguists) but accurate summaries of the field. Mostly agreed upon is the separation of language into distinct layers and modules: phonetics, morpho-syntax, syntax, categorial semantics, lexical semantics, pragmatics. Typically, these respective layers and components are the more deeply understood, the earlier they appear in the preceding list.

The status in linguistics currently appears to be that the basic outlines of phonetics, morphology, morpho-syntax and syntax are well understood [22]. Categorial grammar (a.k.a. Montague Grammar), which studies formal semantics of language, has been a hot research

[7]There appears to be a strong consensus that the innate language capabilities of the human species have evolved under intense evolutionary pressure for better and ever better language capabilities – see [22] for an example. Consequently, our solution to this problem, having been the main target of this evolutionary pressure, is most likely a high-quality one, even if not a perfect solution.

[8]This term is due to P. Bondré-Beil, personal communication.

topic during the last decade or two, with major advances both in the underlying mathematics (logic) and in linguistics, and while research is still ongoing, some fundamental results appear to have crystallized sufficiently, prompting the publication of the "Handbook of Logic and Language" [25] as an overview of the field in 1997. Lexical semantics, on the other hand, is more of an ongoing research area, but the Universal Networking Language [117] project is an interesting experiment in applied lexical semantics that might be able to serve instead.

The area of a formal pragmatics is beginning to be studied based on the new-found improved understanding of the formal semantics of natural language. Speech act and text semantics are a first step towards this wider goal. For now, however, the more informal results on pragmatics from philosophers of language would need to suffice as theoretical background to our study.

4.3.3 Design Issues

Papers discussing the design issues for a common knowledge communication and content markup language are surprisingly rare. Even those who have produced languages for such a purpose tend to ignore the actual language design question and focus more on the architecture of the system within which it is used. Detailed definitions and users' guides for such languages may be available, but design issues are rarely discussed.

In part, the reason for this can be summarized as in this quotation:

> "It's not that they can't see the solution – it's that they can't see the problem." [125]

As an example for this attitude, W. Kuechlin[9] remarked on my proposed research project (developing a language for communication between computer algebra systems) that he didn't see that the "search space" for this problem was large enough to warrant a Ph.D. thesis on it. In Computer Algebra, user languages are deemed a dime a dozen[10], and the fundamental language design flaws that we described for all the "big" CA systems in [2, 3] may well trace back to this cavalier attitude.

[9] Personal Communication, early 1994.

[10] Anthony C. Hearn, personal communication: the syntax of the REDUCE user language is largely irrelevant, and may easily be changed without affecting the underlying system.

Perhaps surprisingly, this cavalier attitude is not restricted to the field of Computer Algebra, where formal logics play little or no role and higher-order functions are dealt with only as necessary and possible. It can be found to almost the same degree in the field of knowledge representation, as can be seen in our analysis of the two versions of the DARPA Knowledge Sharing Effort's Knowledge Interchange Format (KIF) language in Chapter 7.

As far as I know, our paper on the Objectives of OpenMath[7, 1] was for a long time the only paper that attempted an investigation of the overall design issues for such a language. It identified multiple layers and components for such a language, and classified related work based on which layers and components it covers, and how well the layers are separated out conceptually.

Very recently, these early analyses have been deepened and extended in a paper on the design of the MBase system language [126] which lays the foundation of a mathematical formalization of "mathematical vernacular." In [127] the authors investigate the consequences of Paul Wang's "Internet Accessible Mathematical Computing" protocol architecture proposal [102, 103] for a language component that extends OpenMath into a pragmatics domain. Both these new articles take OpenMath as given, however, and work their way on from there.

Instead of discussing the design principles for communication languages, authors that have proposed and implemented interlinguas for symbolic computing systems have instead tended to focus on the system architecture design aspect of communicating and cooperating heterogeneous software systems. The IAMC proposal just mentioned, and the Multi Protocol project from which it grew, are a case in point. This aspect of a problem solving environment, while a very important research topic of its own, is however beyond the scope of this survey.

CHAPTER 5

APPLICATIONS AND IMPLEMENTATIONS

In this chapter, we report on practical experience that we gained implementing and applying content communication languages in symbolic computation systems.

In 1993/1994, we started designing our own such language, dubbed *CALIF* for *Computer ALgebra Interchange Format*, and built a prototype environment for experimenting with such languages in a message-passing-based heterogeneous distributed symbolic computing system [63].

In 1994, our project merged into the OpenMath project [64], and our subsequent implementations and applications concerned different versions of the OpenMath language.

A prototype OpenMath language proposal was distributed in early 1994 [19], and later that year we proceeded to adapt the content communication testbed described above to this language. In January 1995, we used this with S. Vorkoetter of Waterloo Maple to demonstrate a world premiere: the two general-purpose computer algebra systems REDUCE and MapleV cooperating via this language [128].

Simultaneously, we started working with P. Marti and M. Rueher to implement the content communication component of a cooperative real constraint system solver [69]. This involved implementing (in 1994) a message passing component and the OpenMath prototype in two Prolog systems; the REDUCE and MapleV components from our world premiere demonstration were added in 1995.

After a long and involuntary break from the project starting late 1995, during which our contributions to OpenMath and MathML were restricted to occasional theoretical ones, we resumed work on this dissertation again in earnest in 1999, the year in which OpenMath Version 1.0 (which defines the core abstract data structure layer of OpenMath as well as two encodings) was finally released. However, not until the summer of 2000 did we have a

chance to actively implement this new standard, in R. van Engelen's Ctadel problem solving environment for meteorological modeling.

These experiments with content communication, and some of the lessons we learned from them, are the topic of the several sections in the rest of this chapter.

5.1 The Computer Algebra Information Interchange Format Project

Our Computer Algebra Information Interchange Format (CALIF) project, which started in 1993 at the Graduiertenkolleg *Scientific Computing* of the University of Cologne, aimed "to specify and implement open information interchange among computer algebra (CA) systems" [63], meaning "that CA data need to be coded in as nearly universal a Computer Algebra Information interchange Format (CALIF) as we can make it, while still retaining both time and space efficiency, and that data of this kind needs to be moved using a well-established open transport protocol." [63]

5.1.1 A Distributed REDUCE System

In late 1993 we demonstrated a software prototype which implemented a first step in this direction at the conference on the Design and Implementation of Symbolic Computing Systems (DISCO'93) [63]. Several REDUCE CA systems [65] were shown to cooperate on a toy problem using a language that was specific to REDUCE but independent of its implementation base or the hardware it runs on.

The REDUCE Computer Algebra system is perhaps unique in that it has been implemented in several different (and generally binary incompatible) LISP dialects on a staggering number of different hardware and operating system platforms over the years. In addition, it is quite possible to have three or four different, binary-incompatible implementations of REDUCE running on the same hardware (based on, say, PSL [129], CSL [130], Common LISP [131], and Scheme [132]).

The REDUCE system can be made available in so many different ways because it is completely written in a very simple LISP dialect called Standard LISP [133] which is fairly easy to implement either in a separate LISP system (such as a generic Common LISP

or Scheme which would make it available on all platforms these LISP dialects have been implemented on, or a highly architecture-specific LISP implementation such as UT-Lisp [134] for the CDC Cyber 170 series), or which can be implemented directly in a programming language such as C (e.g. CSL) or directly via its own compiler in machine code (such as the highly optimized series of Portable Standard LISP (PSL) based implementations of REDUCE).

But it is not only the compiled code of the REDUCE system that differs dramatically between implementations even on the same architecture. Internal data representations of different REDUCE systems, while compatible on an abstract data structure level of abstraction (lists are lists, symbols are symbols), are in general not identical bit patterns – indeed, different implementations on the same hardware likely have different bit patterns for the same data structure, and the same is true with a vengeance for a single series of implementations such as PSL across hardware platforms, varying greatly with machine data word size (implementations on 32-bit and 64-bit word [135] architectures exist), machine byte order (little endian or big endian), and optimization properties of the machine code architecture (leading to 5-bit or 8-bit tag fields).

Thus, even implementing communication between a PSL-based REDUCE system running on a SUN Sparc architecture on the one hand, and a CSL-based REDUCE system running on a DEC Alpha architecture on the other already provided several of the challenges that the much more general case of a general exchange format for CA systems pose:

- computing hardware differences,
- operating system differences,
- binary incompatibility of internal data structure representations,
- differences in the system-level extension mechanisms.

The first two of these problems had long been addressed by Sun's XDR (External Data Representation) [136] component of Unix RPCs (Remote Procedure Calls) or by ASN.1 (Abstract Syntax Notation) [137] BER (Basic Encoding Rules) [138], but the latter two posed new problems specific to communication between CA systems. For the third, a simple solution existed in the form of data exchange via strings or streams generated by LISP's

READ and PRINT functions, although there were known problems with loss of information when transmitting floating point numbers in their decimal scientific notation representation.[1] The fourth problem, however, was and is a specific problem of symbolic computing systems, which tend to be complex monolithic software systems with idiosyncratic foreign-function-call based system-level software extension mechanisms.

Another class of problems surrounded the question of managing the communication channels between different CA system processes. This was a topic that several efforts parallel to ours (DSP [139], ASAP [61], MP [56], Wolfram Research Inc.'s Mathlink [53] and several others) had addressed by defining a Unix Socket based protocol of their own, and the OpenMath effort which started that year even went so far as to reserve both a UDP and a TCP well-known port number for a prospective OpenMath protocol.

Our REDUCE prototype, by contrast, treated the handling of process management and lower-level inter-process communication as "someone else's problem," as S. Gray phrased it later [128] with reference to [140], and it delegated this aspect to PVM3 [66], which provided cross-platform process management and message passing for simple binary data, and which in turn handed the problem of low-level data conversion (integers, floating-point numbers, strings) to Sun's XDR.

5.1.2 "Toy" Application

At this point, we had looked at the problem from the bottom up, as it were. In order to balance this approach, and to enable a good design, we approached the problem simultaneously top-down through a "toy" application – "toy" in scare-quotes because that toy application included a universal remote procedure call (RPC) mechanism.

The toy application was to enable the running of REDUCE benchmark files in parallel; in particular, the benchmark test for the REDUCE symbolic integration system was run with super-linear speedup[2] using several REDUCE "slave" processes evaluating the integrals and

[1] J.H.Davenport, personal communication.

[2] Speedup was super-linear when ignoring the startup time for the entire system. It was expected because computation times for this particular problem tend to be dominated by garbage collections, which were less frequent in the parallelized version because the total amount of memory available to the total system increased in a linear fashion with the number of slave processes.

REDUCE symbolic integration benchmark with parallelization directives added
parallelization directives for REDUCE top-level command loop
dynamic scheduling
farm of slaves
remote REDUCE procedure call (synchronous and asynchronous)
"slave" REDUCE (simplification server)
send and receive symbolic data structures (s-expressions)
encode / decode symbolic data structures
S-expression buffer management
PVM3 language binding for LISP
PVM3 library

Table 5.1. Layers of first distributed REDUCE prototype

a REDUCE "master" process handing out problems to its "slaves" (with simple dynamic load balancing).

5.1.3 Architecture

The components we added to REDUCE were as shown in Table 5.1. The lowest level in this list, the PVM3 library, was itself highly structured in multiple levels of abstraction, but we consistently treated it as a black box, as "someone else's problem" [140]. Thus, despite being itself riddled with system-specific compilation directives, we treated it as basically system independent.

The PVM3 language binding provided a system independent API to higher levels, but its implementation was necessarily highly system specific, depending as it did on the foreign-function calling mechanisms available in each symbolic computing system implementation. Buffer management, too, provided a system independent API with some system specific optimizations. All the higher levels of abstraction in this list were implemented in a fully system independent manner.

In addition, each level in this list was designed to be largely insulated from both lower and higher levels, in the sense that making changes in specific implementations at a given level would leave the code in neighboring (and other) levels unchanged.

In particular, since experimenting with languages for symbolic communication was an important goal of this implementation, the functions for encoding and decoding the symbolic

data structures were insulated as much as possible from the surrounding levels, and several different encodings were tested.

Note that in this particular instance we were satisfied with transmitting symbolic data structures (S-expressions), since all REDUCE systems use the same abstract data structures for identical purposes. It was not necessary yet to consider the problem of translating to and from a lingua franca for communicating between different computer algebra systems – that was purposely left for the next few steps (see below).

The first encoding we implemented was, of course, that of a string in LISP `print` format. For the purpose of enabling parallel symbolic codes to run (which was our goal), this format had some drawbacks – it was more verbose than strictly necessary (longer strings take longer to transmit), it was "lossy" with respect to floating point data[3] (making it useless for certain applications) and with respect to structure sharing[4] (which can induce exponential growth in memory space requirements if it is missing)[5], and it required the entire LISP string to be transmitted in a single buffer.

A second version replaced the LISP-syntax text string with a simple binary byte-code encoding with a potential for addressing the first and the final two of these concerns, but with an embedded text-string encoding for floating point data still deficient with respect to the second one.

A third version of our encoding was somewhat more complex, utilizing dual buffers (one for a binary byte code representation of symbolic structure data, and one for binary floating point data) in order to provide loss-minimal (and fast) binary floating-point data transmission while taking advantage of the underlying PVM3 (or rather XDR) data conversion capabilities for floating-point data. Basically, a floating-point buffer would be transmitted before a byte-code buffer containing a reference to it, thereby ensuring the availability of numerical data at the time it was needed for parsing on the receiver's end while minimizing the overhead involved in floating-point conversions by fitting as much data into a floating-point buffer as possible.

[3] J. Davenport, priv. comm., *ca.* 1987

[4] That is, the Standard LISP dialect that REDUCE is defined in – unlike Common LISP, for example – does not specify a syntax for labelling and referencing common sub-expressions.

[5] see [1] for an example.

All this had the problem that the encodings were purely ad-hoc (except the LISP text-string one). We therefore investigated standardized binary encodings for their usefulness in supporting the transmission of symbolic data. Clearly, XDR did not qualify as such – what we were looking for was a ready-made encoding for much or all of at least the basic ingredients of symbolic expressions: strings, lists, vectors, and integers, all of which of arbitrary and, in principle, unlimited size and/or nesting depths, names, and floating-point data. In 1994, the only available candidate for this that we could find were the Basic Encoding Rules (BER) for the Abstract Syntax Notation (ASN.1), both ISO international standards (ISO 8824/8825 [137, 138]).

However, there were several drawbacks to ASN.1/BER that limited their usefulness:

- The "size" parameter of an ASN.1 sequence's BER encoding provides its size in bytes, not in number of entries. However, in symbolic expressions the size in bytes of an encoding of a list can only be determined after the entire list contents has been encoded, which means that a symbolic expression would need to be fully encoded before even the first byte can be written to the send buffer. While there are ways around this problem, it points to an incompatibility of BER with fundamental requirements for encodings for symbolic expressions.

- BER encoding and decoding reportedly tended to be comparatively slow if ASN.1 compilers were used to produce the code library.[6]

- Existing tools for converting ASN.1 specifications into efficient BER parsers and generators were hard to come by and even harder to use.

- BER parsers were very hard to implement in non-C environments such as PSL or Prolog, since bit-stream processing was not well supported in most LISP dialects or other symbolic computing languages.

- Import of complex structures into a monolithic LISP or Prolog environment using foreign function calls to a BER library is much more expensive (in performance) than parsing it within the LISP system itself (because the former would require many foreign

[6] Citation can no longer be traced.

function calls, which are expensive). Writing a BER parser from scratch, however, is also a daunting task, even if it is only for one particular ASN.1 expression type.

In the end, we drafted an ASN.1 specification for a possible general-purpose computer algebra interchange format, and used it in presentations up to and including the first OpenMath workshop we attended [17], but never implemented them because the OpenMath effort we joined in 1994 took us in another direction.

5.1.4 Implementation

Our distributed REDUCE pre-prototype was implemented on several different architectures and machines, ranging from SUN Sparc and DEC Alpha workstations to a KSR1 parallel computer, and using both PSL and CSL based versions of REDUCE. Problems with the low-level binary I/O routines under PSL prevented a full implementation of the later binary encodings in the PSL series of REDUCE implementations, but we were able to install the CSL series of implementations on a wide range of different architectures, establishing communication between any pair of such implementations.

Independently and at roughly the same time, Melenk and Neun implemented a PVM3-based communication library for REDUCE, called RR [141], for the PSL-based series of REDUCE implementations that they maintained at the Konrad-Zuse Zentrum in Berlin. It never went beyond beta testing status, however, and only a few beta testers ever used it.[7]

Known limitations of our own implementations included:

- LISP's so-called *gensyms* were not handled properly
  (A closely related problem was addressed later in the early Prolog implementations of the first OpenMath prototype (see below)).

- structure sharing information was not preserved
  (There existed well-known algorithms for handling the linearization of DAGs or even graph structures, so that we didn't consider this a problem that we needed to address at that point since efficient solutions were known to exist.)

[7] W. Neun, personal communication.

- lack of protection against interference between different processes using the same simplification server as an RPC service with state
  (Implementation of multi-session handling within REDUCE would have required an implementation of thread-like controls in that system. Since our focus was to be on the communication language, this was clearly beyond the scope of our investigation. R. Sutor demonstrated session handling for a "Saturn" (later called TEXexplorer) user interface to the Axiom/Scratchpad computer algebra system backend at a poster session at ISSAC'95, showing both that this was a difficult and a solvable problem.)

- most of all lack of support for CA systems other than REDUCE.

5.1.5 Early Lessons

Despite its many limitations, both its successes and its concrete problems already provided some valuable lessons.

1. *Separate data structure and mathematical content language layers:* Importing or exporting a data structure to a common language is rather straight-forward, and already useful in intra-CA but inter-architecture communications. Semantic layer translations conceptually require CA algorithms, and are conceptually quite distinct.

2. *Focus on language, not protocol:* Inter-process communication and control protocols (PVM, MPI, distributed shared memory, COM, etc.) are a research area all of its own, as are parallel processing communication patterns (hypercubes, hubs, farms, grids, etc.) and parallel programming paradigms (message passing, virtual shared memory, etc.) and languages (PVM, MPI, Linda, etc.). Any and all of these need to be supported in the interest of research and development on parallel and distributed symbolic computing. The problem of devising a lingua franca for communication in a heterogeneous network of CA systems is largely orthogonal to all these research areas, and thus needs to be treated this way.

3. *Support multiple encodings:* There is no single encoding that meets all requirements. For many purposes, including debugging, a simple text encoding like the LISP "print" one used as an initial quick-and-dirty solution is perfectly fine, and easy to implement

in almost any environment. Certain binary encodings are required for more advanced uses, but we saw that some systems may not be able to support them due to some bug or other in the system, or may simply have no interest in supporting them if a simple text encoding suffices. Thus, a CA communication language needs to support a variety of encodings.

4. *Separate data structure and encoding language layers:* It follows from lesson 3 that the data structure and encoding layers of the CA communication language need to be separated. After all, it may be necessary to support multiple encodings, but there does not appear to be a need for more than a single abstract data structure, since the history of LISP shows that a fairly small but well-chosen "universal" set of primitives is sufficient to support the representation of any mathematical structure that a computer would be able to handle.

These lessons led to our first significant contribution to this field, the recommendation of a three-tier language architecture (see Figure 5.1). As the core middle tier we proposed an abstract data structure layer with a well-defined common set of abstract data-structures defined for all applications. The lower tier would branch out into several encodings (LISP-like text encoding, binary byte-code encoding, etc.) that linearize the data structure in a loss-less manner. The upper tier would branch out into definitions of representations for a plethora of mathematical concepts using the data structures of the middle tier.

This is essentially the view taken by OpenMath today [10].[8]

5.2 First Full Prototypes

5.2.1 An OpenMath Prototype

In 1994, Stefan Vorkoetter of Waterloo Maple posted a draft version 0.1 for an OpenMath language [19]. We proceeded to add this encoding to our existing list of encodings that the parallel REDUCE prototype was able to "speak."

The architecture that we described in the previous section proved remarkably robust. Encodings had been purely on the data structure level so far, and this was the first time that

[8]Subsequent experience suggested a refinement to this layering, but that extra twist did not make it into the OpenMath specification.

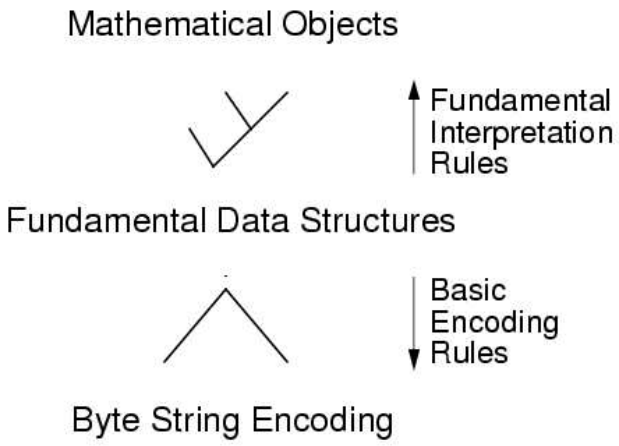

Figure 5.1. OpenMath Layers proposed at the Second OpenMath Workshop

semantic translations had to be incorporated in the encoding and decoding process, but the software architecture needed only small changes to accommodate this.

Importing an OpenMath 0.1 object into REDUCE was implemented as a two-stage process.

First, the regular "decode" function in the existing architecture was replaced by a syntactic data structure import for REDUCE – generating LISP lists, integers, strings, and symbols in a way that closely mirrored the LISPish syntax of OpenMath 0.1. However, this was a purely syntactic import: names in the OpenMath expression were imported to symbols in what amounted to a clean name space (as opposed to being `intern`ed as-is). This way, an internal representation of an OpenMath object (as opposed to its direct REDUCE translation) was created in the REDUCE heap by the decode (or `read`) function.

In the second stage, the semantic translation of the OpenMath Object to its REDUCE equivalent was then triggered by the "simplify" phase of the "receive–simplify–send" loop of the simplification server. This was implemented in the form of a set of REDUCE simplification rules for OpenMath Objects which would fire before the "real" REDUCE-

specific simplification rules could be triggered for the purpose of performing the remote procedure call.[9]

Exporting an OpenMath Object again followed this bipartite pattern of separating semantic from syntactic processing. First, a set of REDUCE simplification rules was again used to transform a REDUCE internal representation of a mathematical object into a REDUCE internal representation of its equivalent OpenMath object data structure, which would then later be transformed syntactically to an OpenMath 1.0 encoding, in the same place of the architecture as before.

The experience we gained here reinforced the lessons previously learned regarding a need to separate syntactic and semantic layers of abstraction of the CA communication language, not just in the abstract, but also in the concrete programming of OpenMath data I/O. It also vindicated the decision to design the software such that the encoding was a pluggable parameter, and thus reinforced the lesson on having a separate encoding layer as well.

5.2.2 A World Premiere

In parallel efforts, several others implemented the draft OpenMath communication language for other CA systems, among them S. Vorkoetter of Maplesoft, who had written the draft specification and had implemented it in MapleV. In the week before the 3^{rd} OpenMath Workshop [128], during a visit at our offices at the University of Cologne, we used our two respective implementations of the OpenMath prototype language for REDUCE and MapleV and produced the first prototype of REDUCE and MapleV communicating with each other using the OpenMath prototype language.

In less than a week, we managed to implement in MapleV a PVM3 language binding along the lines we had specified a year earlier for LISP languages [63], and were able to use MapleV "slaves" to run the "parallelized" REDUCE symbolic integration benchmark test that we had previously developed. Except for bug fixes in the syntactic analysis of the prototype OpenMath language implementations, and changes to the system parameters

[9]This rule-based semantic import was chosen for generality and ease of implementation. Since these rules were mostly rather trivial, it was clear that a special-purpose tree-traversal implementation would have been far more efficient. However, since it was also quite obvious how this would have been done (indeed, an automatic compilation process might have yielded such code), there was no particular merit in actually going through the motions of such an optimization until OpenMath itself stabilized.

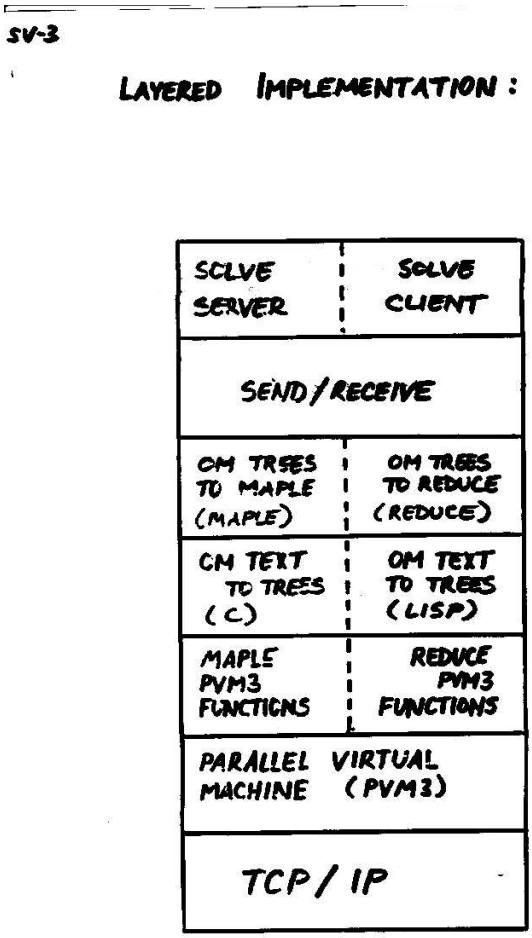

Figure 5.2. Layered Implementation of OpenMath prototype in REDUCE and MapleV

for starting the OpenMath-enabled MapleV processes instead of REDUCE processes, our REDUCE implementation needed no changes – another indication that we were on the right track with our architecture.

Consequently, S. Vorkoetter presented the common architecture of our first prototype implementation of a heterogeneous cooperative system of general-purpose CA systems communicating via the *lingua franca* OpenMath 0.1 at the OpenMath Workshop a few days later [128] (see Figure 5.2).

5.2.3 More Lessons

Perhaps the first thing we noticed when we let REDUCE and MapleV "talk" to each other like this was that even though they both clearly spoke the same language (the OpenMath prototype), they both just as clearly spoke it with a different "accent" that mirrored each system's internal representation of the information exchanged via this *lingua franca*. MapleV, for example, would use an expression saying $2^{1/2}$ while REDUCE would use $\sqrt{2}$, and REDUCE would perhaps use a complex expression involving the error function where MapleV would give an answer in terms of the γ function that REDUCE did not have any built-in knowledge of.

Thus, it became quite apparent that one computer algebra system conversing with another would always be required to do some work to massage information it receives into its own internal representation, unless that other system happens to use a compatible internal representation itself. It also became clear that controlling for shared vocabulary among communication partners is a crucial ingredient of communication between intelligent "agents" such as these. For both these reasons, we concluded that "implementations will not be trivial for CASs."

We also found that in real life, so to speak, the functions, predicates, and quantifiers that OpenMath defined were not enough; commands like "solve" were necessary, too. In addition, we noticed that even in the simple examples we ran the vocabulary defined by the prototype OpenMath was not comprehensive (it lacked "root-of", for example, an example of a generalized quantifier with qualitatively different implementation in the two CASs we had joined). Extensibility thus played a crucial role in a language like OpenMath, as we simultaneously noted in our Objectives report to the workshop [6].

5.3 An Application: A Cooperative Multi-Solver Constraint Resolution System

The first OpenMath prototype discussed in the previous section actually formed one of the components of a concrete application that, six months earlier during a visit at INRIA Sophia-Antipolis, we had been invited to help implement with P. Marti and M. Rueher at the graduate school of computer science (ESSI) at the University of Nice, France. Indeed,

we had succeeded (with caveats) in implementing communication via the same OpenMath prototype and PVM3 between two different Prolog systems, Prolog III and Delphia Prolog, before demonstrating cooperation between MapleV and REDUCE.[10]

The problem domain this research group was concerned with was called Constraint Logic Programming (Prolog III was a system of this kind), which adds to pure logic programming (as implemented for example in Delphia Prolog) a notion of constraints (inequalities over the real numbers) that further limit the range of values admissible for logic variables.

5.3.1 The Problem

The particular subproblem that Marti and Rueher's particular system was to address was that of finding solutions to constraint systems, that is simultaneous solutions to sets of equations and inequalities over the reals. Full solutions to this problems were unknown, and exact solutions to the problem within a significant subdomain using Cylindrical Algebraic Decomposition were known to have unacceptable computational time complexity. Therefore, the actual task given was simplified to finding hyper-intervals that are guaranteed to contain all possible solutions to the input set of inequalities over the reals. Note that no limitations were in principle given for the left and right-hand sides of these inequalities – Marti and Rueher were interested in non-linear systems.

As a twist on the more usual approach of defining the problem as requiring as output a *single* hyper-interval containing all solutions, Marti and Rueher generalized it to allowing as output *a finite set of* hyper-intervals. In the case of non-linear constraint systems, this was known to provide a chance to provide much closer approximations to a solution, as in the trivial example $\{x^2 > 100, x^2 < 101\}$, where a single-interval solution might read $\left]-\sqrt{101}, \sqrt{101}\right[$, while a multi-interval solution could improve on this dramatically by returning instead $\left]-\sqrt{101}, -10\right[\cup \left]10, \sqrt{101}\right[$.

Given the complexity of the full problem, and given that the problem was concerned with finding good approximations for the actual solution (and with having a good shot at identifying unsolvable problems), a large number of solvers existed that handled sub-spaces of the full problem set, with varying performance characteristics. It was also understood that distributing the single-interval-solution variant of the problem and letting the single-

[10] Needless to say, these implementations also followed the architecture described previously.

interval-solution solvers cooperate in its solution could provide a tighter approximation than any single one of them might produce on its own.

5.3.2 The Proposed Solution

Marti and Rueher had theoretically extended this cooperative constraint system solving paradigm to the case of multiple-interval solutions, provided that at most one of the sub-solvers was capable of generating a branch-point in the solution path (i.e. capable of generating disjunctive partial solutions) [69].

To verify their theory in practice, it was decided to implement a cooperative constraint solver consisting of several solvers that each could handle a different sub-domain of the full problem set, and of a coordinator process implementing the algorithms they had developed.

The tools

The following are the off-the-shelf solvers used in the research prototype:

Simplex: The simplex algorithm takes as input a linear constraint system (i.e., the left- and right-hand sides of inequalities are limited to terms that are linear in the free variables), and it returns a single hyper-interval that is guaranteed to contain all solutions, should they exist. If it returns an empty interval, it guarantees that no solution to the linear real constraint system exists.

Interval Propagation: The interval propagation algorithm takes as input a general real constraint system (with evaluable real functions usually taken from a small set including exponential and trigonometric functions) and returns a single hyper-interval containing all solutions, should any exist. It uses interval arithmetic internally.

Groebner Basis Computation: This class of algorithms can be used to solve exactly a certain class of polynomial systems of equations (but not inequalities).

Of these, the simplex algorithm is generally the fastest, but with a severely restricted input domain of systems of linear real inequalities. The particular implementation used required these to be of the form $a_1x_1 + \ldots + a_nx_n \sim a_0$ where $\sim \in \{=, \neq, <, \leq, >, \geq\}$ and the a_i, $i = 0 \ldots n$, are numbers of about 15 decimal digits. It returns partial solutions of the

form $x_i \sim b_i$ that it discovers to hold (with b_i again roughly 15-digit numbers); altogether, it returns a single hyper-interval as a solution, which may be infinite in some or all dimensions. This solver is incremental, that is it can add or remove real linear constraints to or from its current state, and update its solution every time it does so.

The interval propagation algorithm is in principle able to handle the entire problem domain, taking non-linear systems of inequalities with left- and right-hand sides of inequalities being arbitrary compositions of real functions (addition, subtraction, multiplication, powers, exponential function, and logarithm at a minimum). Its output is of the same kind as that of the simplex algorithm, effectively returning a single hyper-interval (potentially infinite) as a result. It is significantly slower than the simplex algorithm in general. In addition, it has serious convergence problems with systems where variables appear more than once, for example.[11] It is incremental in the same sense as the simplex algorithm.

These two solvers were basically implementations of essentially numerical algorithms. The Groebner Solver, by contrast, implements a purely symbolic algorithm. Its input is a system of algebraic equations (that is to say equations with left- and right-hand sides that are polynomials over the rationals), and its output is a symbolic description of a set of algebraic solutions to the system (if exact solutions exist). Note that it processes equations rather than inequalities, and that it handles truly non-linear constraints, although no transcendental functions are allowed. In addition, it does not handle approximate constraints of this sort (i.e. polynomials with floating point number coefficients) – only exact rational numbers are allowed (but these can have an arbitrary number of digits). Although the Groebner Basis algorithm used in this Solver has been proven to have a worst-case super-exponential space complexity (i.e. there is a class of input problems the size of whose output solutions grows super-exponentially in the input problem size), it tends to have a reasonable performance on many problems.[12] Unlike the other solvers, Groebner algorithms are not designed for incremental use, which slows it down even more since adding or removing a constraint generally means re-computing the solution of the system from scratch.

[11] Ph. Marti, personal communication. See also [74].

[12] A.C.Hearn, one of the founding fathers of the field of computer algebra, is on record with a conjecture that all physically derived (as opposed to mathematically 'concocted') problem classes for the Groebner basis algorithm are well-behaved wrt. their time and space complexities.

Note that all of these tools take mathematical formulas as input, but only the Groebner basis computation (or rather, the polynomial-systems-of-equations solver) returns mathematical formulas as solutions, too. Also, the Groebner solver is (by design) the only one that returns *sets* of solutions rather than a *single* solution.

Orchestrating a cooperative solver

Marti and Rueher then proposed to realize a cooperative constraint system solver from these components[13] by adding a controller to orchestrate their cooperation. This process, called the Monitor, had a variety of tasks to perform:

- Transform the problem to match the input specifications of all target solvers.
- Broadcast the problem to the component solvers.
- Asynchronously receive answers from all solvers. Depending on the class of answer received from a solver, the Monitor needs to react in different ways:

Case 1. The solver replies "success": It notifies the Monitor that the information it received is consistent with the solver's current state, but it was not able to reduce the problem further. Nothing needs to be done by the Monitor except noting this reply.

Case 2. The solver replies "fail": It notifies the Monitor that the information it received has been determined to be inconsistent. The Monitor needs to notify all solvers that they can abort the current branch of computation and should backtrack to the previous choice-point. (Once all have backtracked and notified the Monitor, the Monitor needs to backtrack within its own internal representation of the problem description.)

Case 3a. The solver replies with a new constraint of the form $x_{i_1} \sim a_{i_1}, \ldots, x_{i_m} \sim a_{i_m}$, where $\sim \in \{=, \neq, <, \leq, >, \geq\}$: It notifies the Monitor that it has managed to narrow down some of the dimensions of the problem to a certain hyper-interval. The Monitor needs to inform the other solvers of this additional information, and needs to add it to its own description of the problem being solved.

[13] Of course the idea was that it would be open to inclusion of more solvers into the cooperation.

Case 3b. The solver replies with a set of new constraints: The Groebner Solver (the only one that can do this) notifies the Monitor that it has found a set of solutions to the polynomial system of equations it was given. When it processes these, the Monitor needs to let all solvers synchronize before asking them to mark a choice-point, i.e. a point in their computations that they can backtrack to. Then, one by one, it processes the elements of the set it got from the Groebner solver by sending an appropriately transformed version to each of the other solvers to open up a new branch of the computation. The next solution in the list will be processed in the same way once the whole system has backtracked to the choice-point marked for the previous element of the list.

- When all solvers have replied "success" on the same branch of the problem, the Monitor reports the current set of new constraints of the form $x_i \sim a_i$ to the user interface, and backtracks to the previous choice point if one exists in order to find the next solution. If no choice point exists, it cleans up and finishes.

Architecture

Marti and Rueher therefore propose the following architecture for the cooperative solver:

The Monitor implements the flow-of-control of the cooperative solver. It sends constraints to its solvers, and receives their results. It keeps track of the global state of the computation, synchronizing the branches of the global search space that the different solvers are processing. By design, the monitor does not need to "understand" anything of the specific constraint system domain. Rather, it is modeled as a message-passing based general-purpose mechanism for coordinating constraint solving.

The Solvers are the off-the-shelf software systems capable of handling aspects of the real constraint system solving problem.

Interface Modules have the task of fitting an off-the-shelf solver to the global cooperation model of the Monitor. Depending on the specific Solver to be fitted, this may involve:

Input Filtering: extracting the information the Solver can handle from the stream of constraints sent by the monitor. As an example, the Simplex Interface Module

withholds a non-linear polynomial equation from its Solver, but passes on any linear constraint.

Input Transformation: pre-digesting the input constraints sent by the monitor. As an example, the Simplex Interface Module needs to perform constant folding on linear constraints before handing them to the Simplex Solver if the linear constraint involves symbolic constants such as $\sqrt{2}$.

Input Translation: translating from the standard language used within the cooperative system to the input language of the Solver, unless the Solver understands the standard language.

Constraint Store Book-Keeping: If the Solver is not incremental, its Interface Module needs to emulate an incremental solver by requesting the result for the full set of constraints that are relevant to the current branch of the computation.

Protocol Gatewaying: The autonomous Solver may not provide a full message passing interface based on the message passing library used in the cooperative solver (i.e., between the Monitor and the Interface Modules). Often, only *stdin* and *stdout* are available as interfaces. The Interface Module has to serve as a gateway between these kinds of protocols if necessary.

Output Translation: translating from the output language used by the Solver to the language used within the cooperative system.

Output Transformation: post-processing the result returned by the solver. As an example, $X = 1.23$ as returned by the Solver may actually "mean" $1.225 \leq X \leq 1.235$ (i.e., the Solver's algorithm only guarantees that the latter holds when it actually returns the former as an answer). A second example that would involve more complex transformations is common in results returned by a Groebner Solver: here, results often need to be phrased in terms of the roots of (univariate) polynomials because no closed-form solution involving n'th roots may exist. In such a case, a single formula may in fact encode the collection of all possible ways of instantiating it with specific roots of the polynomials involved. In terms of the cooperative solver, this needs to be re-phrased as a disjunction of all these possible instantiations.

Output Filtering: making sure that only new and relevant information from the result returned by the Solver is returned to the Monitor. Examples are:

- Only narrowed-down interval solutions are relevant; result intervals contained in input intervals are not reported.
- Only real solutions are relevant; results with non-zero imaginary parts are filtered out.

Note: In the actual implementation, some of these tasks were done in the Monitor process, and some were done by driver code in the solver itself. One lesson that we learned in this project was that the division of tasks as given here is more suitable to the problem than the originally implemented one.

Implementation

The Monitor, the User Interface, and the Interface Modules were implemented as Delphia Prolog processes with a PVM3 library bound in. The Simplex Solver was an off-the-shelf commercial package running under Delphia Prolog. The Interval Propagation Solver was a commercial package running in the Prolog III Constraint Logic Programming system. The Groebner Solver was implemented as a MapleV process with the PVM3 and Prototype OpenMath code by Stefan Vorkoetter built in,[14] the same system that we demonstrated at the 3^{rd} OpenMath Workshop.

In order to support some of the complex transformations required in the Interface Modules or the Monitor, all of which were implemented in a Prolog system with no built-in "big integer" arithmetic, we added to this architecture a general-purpose **Algebraic Computing Module** that acted as a computer algebra server augmented with functions that implemented the algebraic transformations required by some of the servers or their interfaces. This module was implemented as a REDUCE process with PVM3 communication library and OpenMath interface built in, the same as was demonstrated together with the MapleV process we were using.[15]

[14]We thank Waterloo Maple Software Inc. for providing this crucial component of the system.

[15]We gratefully acknowledge the support of Codemist Ltd. and ZIB Berlin for letting us use their REDUCE implementations for this project.

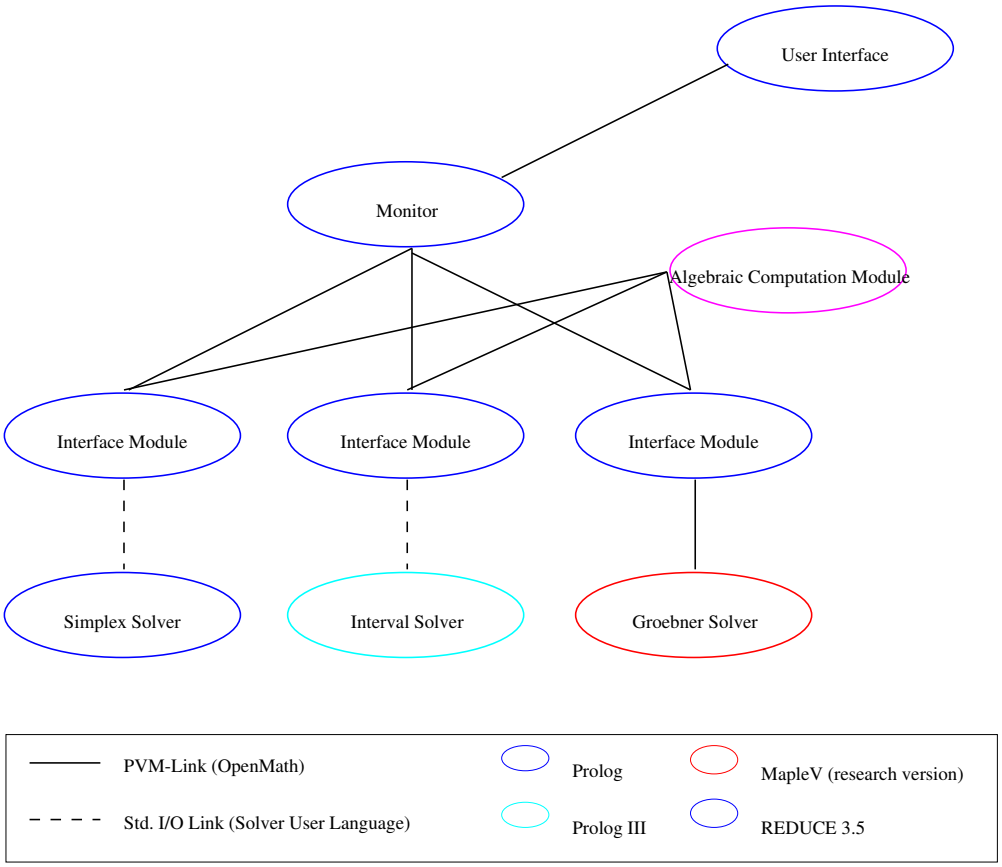

Figure 5.3. Cooperative Solver Processes and Communication Pattern

Figure 5.3 shows the processes making up the cooperative solver, and the pattern of communication between these processes.

5.3.3 Lessons Learned

In many ways, the implementation of this first application of OpenMath reconfirmed the lessons learned earlier. The previously built components that comprised the MapleV-REDUCE prototype we demonstrated with S. Vorkoetter at the 3rd OpenMath Workshop were reused in the application with very little change.

Some interesting new aspects came up when comparing the Prolog implementation of OpenMath communication with the previous implementations in systems with more of a

LISPish functional flavor. Adding PVM3 support to the core Prolog system used in the application was mostly fairly straightforward.[16] The separation of the OpenMath import and export routines into a syntactic and a semantic phase was reconfirmed. The syntactic import into a Prolog data-structure was done using a lexical analysis compiler and a formal syntax specification for the OpenMath prototype language we were then using. It produced a term structure that described an OpenMath object with symbols that were unique in the Prolog name space to the OpenMath language, just as we had done before. Just as previously, too, the internal representation of mathematical formulas (including the vocabulary for common mathematical concepts) was different from that of the OpenMath prototype – in this case, P. Marti used his own representation of mathematical formulas as Prolog terms in all the core processes of the system (the user interface, the Monitor, and the Interface Modules).

However, there were some interesting differences.

The term structure produced from the syntactic import was actually a pure data structure with no inherent meaning to Prolog. This distinguishes it sharply from both the previous REDUCE and MapleV implementations that chose data structures on import that represented the same mathematical structure as that intended by the OpenMath language, namely nested applications with heads applied to arguments.

While names, heads, and arguments mapped fairly well into the structure of Prolog terms, the Prolog system we dealt with did not handle some of the basic data types of OpenMath directly. There were no arbitrary size integers; the handling of floating point numbers was difficult; and variable names do not work well in Prolog at all because Prolog variables are by design anonymous instead of named as in the more LISPish environments we used before. The first two problems were straightforward to solve – by using strings wrapped with special headers as a representation, and by leaving the computations with these data types to the Algebraic Computing Module. The latter problem was more serious, and required keeping a table of variable names and their anonymous variable meanings while receiving, processing, and replying to a message from outside, or while sending, waiting for a response to, and receiving the response for a message originated in the process. This made

[16]The Prolog PVM3 code was a joint effort with P. Marti. The PVM3 Prolog API was based on the one that we had produced earlier for LISP [63].

it necessary to restrict oneself to fairly simple RPC-style communication, and limited the amount of asynchronous parallelism that could be supported.

Thus, we learned an important new lesson when trying to teach a Prolog system to speak OpenMath, namely, that there are really two separate components to the semantic layer of OpenMath. The first of these two components is the one that concerns the combinatorial semantics of the constructors and simple atoms of the language, i.e. which defines semantic rules of composition. In the case of the OpenMath prototype language we were using this meant, for example:

- The meaning of an OpenMath 0.1 expression of the form *(a b c)* is that of an application of the object represented by a to two arguments, viz. the object represented by b as first argument and the object represented by c as second argument.
- Certain patterns of strings stand for OpenMath symbols whose meanings can be looked up in the OpenMath prototype definition or some suitable software realization of that lookup table.
- Variables in OpenMath are represented as symbols that do not have a meaning attached.
- Certain strings in an OpenMath expression stand for mathematical integers. Integers can have an arbitrary number of digits.
- etc.

The second component of the semantic layer of OpenMath, by contrast, consists of definitions of the symbols used in OpenMath, or patterns of "phrases" in which they occur, for example:

`*` is a symbol that means multiplication, i.e. which stands for an n-ary function that, applied to its arguments, returns the product of its arguments.

`pow` means powering, i.e. a binary function that, applied to its two arguments, say, a and b, returns a^b, i.e. a to the power of b.

`(pow` x i`)` with i an OpenMath representation of a positive integer is a phrase that means i-times repeated multiplication of x with itself, i.e. $\Pi_{k=1}^{i} x$. (This definition is for cases where a general powering is not defined for x, but multiplication on x is.)

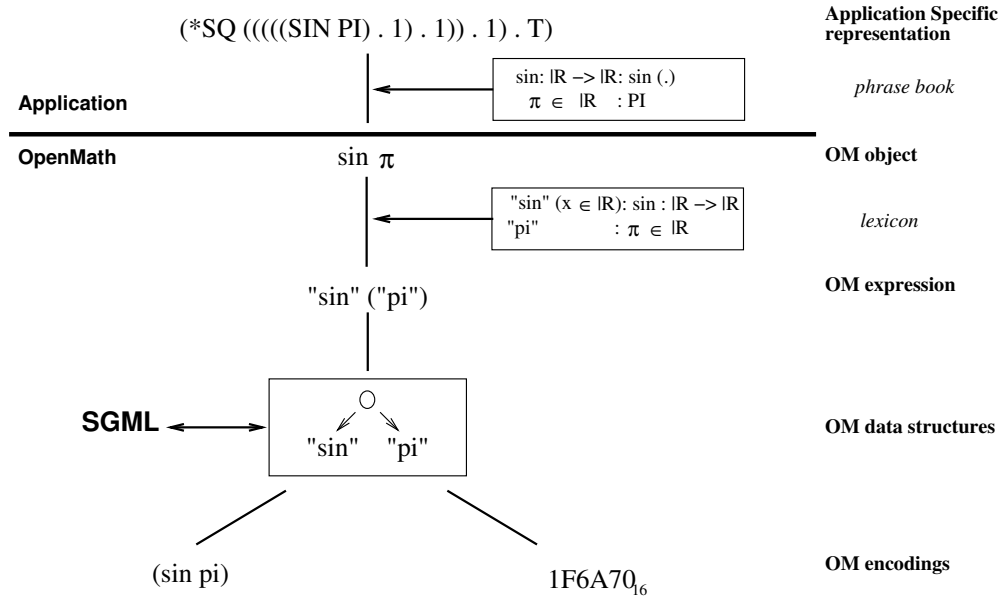

Figure 5.4. Proposal for OpenMath Language Layers from OpenMath Objectives Paper

This refinement shown in Figure 5.4 to our original proposal for an OpenMath language layering shown in Figure 5.1 was formally proposed to the OpenMath community at the same time that we presented the first MapleV-REDUCE cooperative prototype. It was later the core of the document known as the OpenMath Objectives, the first public version of which was distributed at that workshop, and the final and official version of which was first handed out at our ISSAC'95 poster on the "Objectives of OpenMath" and later published as [1].

5.3.4 Practical Problems

Trustworthiness of component systems

The cooperation mechanism proposed by Marti and Rueher relied on the component Solvers to be trustworthy in their answers. It therefore came as something of a surprise that some of our first tests produced a wrong result when all debugging seemed to indicate that the orchestration code worked just fine.

Given the implementation of the core collaborative environment on top of the PVM3 message passing library, we had the ability to visualize the message exchange patterns (using XPVM) and to intercept and trace the messages being exchanged between the Interface Modules and the Monitor. In addition, each process could be traced in its own terminal window due to the decoupling of message passing and standard I/O.

By thus tracking the messages exchanged, we found to our surprise that one of the Solvers, the Interval Propagation Solver, returned incorrect results for systems that involved cubic polynomials: it was apparently not as trustworthy as had been assumed.

As we saw in Chapter 4, in the then-fledgling field of research into providing automatic theorem proving systems with computer algebra system support, this had been a major stumbling block from the start. CA systems in general can only give a best-effort guarantee. Their answers cannot be relied on 100%, either because the algorithms they use have an uncertainty component (e.g. a probabilistic primality test) or because their particular implementation is hardly ever verified for correctness. This is a problem, of course, in a setting where the results returned by the CA system is to be used in a mathematical proof, and various approaches to the reliability problem have been proposed and analyzed in the literature.

However, it turned out in our case[17] that the Interval Propagation Solver could be run in several different modes, one of which was "safe but slow" and able to handle those cases that had caused us problems when using the default "quicker and dirtier" mode. We were thus relieved of the necessity to deal with the much thornier problems that our colleagues who were joining CA and ATP systems had no choice but to confront.

Avoiding misunderstandings

And yet, we could not completely avoid this problem. Recall that we were interested only in working in the domain of real numbers. MapleV's implementation of the Groebner Solver (like many others of its kind) insisted on working in the complex domain, however. While not technically "wrong," it therefore nevertheless frequently "misunderstood" the questions that were intended ("can you solve this set of real polynomial equations over the reals?"),

[17] Ph. Marti, personal communication.

and the MapleV interface had to be programmed so as to filter out those "useless" answers that returned solutions with non-zero imaginary parts.

Similarly, the Simplex Solver theoretically was able to solve any (reasonably sized) system of linear inequalities. In terms of symbolic mathematical formulas, given certain reasonable assumptions, its allowable input set was thus mathematically characterizable as follows:

1. A *system of real linear inequalities* is a finite sequence of *real linear inequalities*.
2. A *real linear inequality* has the form *lhs rel rhs*, where *lhs* and *rhs* are *real linear terms* and *rel* is one of the inequality relations "equal," "not equal," "greater than," "greater than or equal," "less than," or "less than or equal."
3. A *real linear term* is either
   (a) a finite sum of *real linear terms*, or
   (b) a *real variable*, or
   (c) the product of a *real constant* and a *real linear term*.
4. A *real constant* is either
   (a) a floating point number,
   (b) an integer,
   (c) a rational number,
   (d) a named real number (such as e for the base of the natural logarithm),
   (e) or a real function applied to a *real constant*.[18]

In practice, however, its input set was restricted to the set characterized by:

1. A *system of real linear inequalities* is a finite sequence of *real linear inequalities*.
2. A *real linear inequality* has the form *lhs rel rhs*, where *rhs* is a *real linear term*, *lhs* is a *real constant*, and *rel* is one of the inequality relations "equal", "not equal", greater than, greater than or equal, less than, or less than or equal.

[18]Certain restrictions to this rule need to be made in practice, since the square root function applied to the integer -1 does not return a real constant, for example.

3. A *real linear term* is a finite sum of *real linear monomials*.

4. A *real linear monomial* has the form *coeff* * *var*, where *coeff* is a *real constant* and *var* is a real variable.

5. A *real constant* is a floating point number with up to 15 decimal digits mantissa.

Since one of the solvers (namely MapleV) would frequently produce results (e.g. $y = -\sqrt{3}x + \sqrt[4]{2}$) that are of the former, more general, form, but not of the latter, more restricted, form, it was necessary, on the one hand, to have an algorithm that *recognized* linear real constraints in their more general form, and on the other hand, to have an algorithm that coerced a constraint given in this more general form into one that conformed to the more restricted form. Since this involved turning symbolic constants into numeric ones (*a.k.a.* constant folding), our application called for algorithms that would guarantee the precision of the floating point numbers generated in this process.

REDUCE[65] (with its Bigfloat package) was easily programmed to do the required combination of simplification and guaranteed-precision constant folding where an implementation of this needed capability in the Prolog system we used would have been prohibitively expensive in terms of manpower required. It was therefore only natural to add the Algebraic Computation Module, which implemented this and other computer algebra functions, and to use it from the Prolog modules via remote procedure calls using the same communication mechanisms that we already utilized in communications between the core components of the cooperative constraint system solver.

Ontologies and phrase books

Another question we encountered was this: What is a good way to represent the concept of a real constraint system? Computer algebra systems usually return the result of solving a set of polynomial equations as a set of sets of (simpler) polynomial equations, but was this the representation that we wanted?

This question was one that the OpenMath community would now call one of Content Dictionary design, and that a related field would term a question in the design of ontologies.

In this instance, we took the stance that it is the conceptual structure of the application that determines the structure of the ontology. More concretely, the theoretical foundations

developed by Marti and Rueher underlying the application prototype that I was helping Ph. Marti develop viewed a real constraint system as the disjunction of conjunctions of real inequalities rather than as a set of sets of inequalities. Therefore, that's how we decided we would represent a constraint system in the OpenMath prototype language we used.

Unfortunately, this complicated matters considerably in one of the systems we used, namely, REDUCE. This computer algebra system had a very restricted sense of the Boolean connectives which it applied mercilessly, and in our case most inappropriately. The literal translation of a system of constraints represented in the conceptually proper way in OpenMath simply did not work – interpretation of Boolean connectives by REDUCE's simplifier had to be avoided at all costs, and an application-specific translation of such Boolean expressions into sets of sets of inequalities (and back) was required to handle the problem.

A lesson that we can learn from this is that one cannot rely on two symbolic computing systems to use the same *mathematical* structures to represent a concept (let alone the same *data* structures). When communicating mathematical knowledge between software agents, therefore, a general-purpose "phrase book," as OpenMath now calls it, translating common mathematical concepts between their symbolic-computing-system-internal representations on the one hand and their canonical external representations in a knowledge communication language like OpenMath as provided by the makers of the symbolic computing system is not enough: these would translate Boolean expressions to Boolean expressions, and sets of sets to sets of sets. There also needs to be a user-level mechanism for extending the "phrase book" with application-specific translations: one that allows a user to determine that under certain circumstances a set of sets needs to be translated into a disjunction of conjunctions, or vice versa.

Indeed, we should learn that it is a good design principle for ontologies to annotate a mathematical representation of a concept with a designation for that concept itself – to explicitly say that "this set of sets of inequalities represents a constraint system," for example. Again, this argues for user-extensibility of the phrase book, but in this case indirectly because it is the ontology that the phrase book translates that needs to be user-extensible, too.

5.4 Teaching Ctadel to Speak OpenMath 1.0

During the summer of 2000, we implemented an OpenMath 1.0 import and export mechanism for R. van Engelen's Ctadel [142, 143] model compiler, a small, special-purpose computer algebra system used for compiling mathematical specifications of a certain class of meteorological models into high-quality Fortran code for a wide range of computer architectures.

As we argued in considerable detail in [5], adding OpenMath symbolic communication capabilities to Ctadel would eventually allow building a problem-solving environment for climate studies with Ctadel at its core with which a large and extensible set of other tools would collaborate. As an example, we envisioned communicating with a library of formal descriptions of numerical algorithms, e.g. the one accompanying the NAG library of numerical code, to enable Ctadel to optimize code that included calls to algorithms that it describes.

At this point in time, OpenMath 1.0 had finally been released officially in a stable definition [10]. The standard document included at its core the specification of an abstract data type for OpenMath Objects along with an informal declaration of the intended meaning of its ingredients, and as separate appendices the specification of two OpenMath encodings – one based on XML [37], the other a byte-code encoding. The basic vocabulary ("Content Dictionaries") for use in OpenMath objects was undergoing a final revision before official publication, and we had agreed to act as external reviewers.

OpenMath 1.0 was quite a bit more complex than the OpenMath prototype language that we had used in the previous projects. Here are some of the things that had been added in the intervening years:

- More constructors: Unlike the prototype, whose single ingredient for composing objects from smaller ones had been application, the more modern OpenMath 1.0 defines four constructors: Application, Binding, Attribution, and Error Objects. Of these, the latter two are meant to be semantically neutral, but Binding Objects play a crucial role in interpretations of OpenMath Objects [3].

- More Basic Object types: In addition to the symbols, integers, strings, and decimal floating point numbers of the prototype, OpenMath 1.0 distinguishes variables from

symbols, adds namespace information ("Content Dictionary") to a symbol name, and makes provisions for hexadecimal representations of IEEE-formatted floating point numbers.

- XML syntax: Whereas the prototype used a very simple LISPish syntax, true to its having only the single application constructor, the "human readable" encoding for OpenMath 1.0 was defined so as to be compatible with XML.[19]

- Formalized Content Dictionary: The prototype included a small set of predefined symbols, but OpenMath 1.0 has no predefined symbols whatsoever. Names for mathematical concepts are instead grouped together into named "Content Dictionaries" (CDs), and the OpenMath 1.0 standard document defines an XML-based format for CD files.[20]

- Formalized CD Groups: In addition, collections of related Content Dictionaries can be given a formal name by declaring a CD Group. Again, a formal specification for a CD Group file format is given in the standard. The CD group that we were reviewing that summer was the core one, called "MathML" because it provided compatibility with MathML-Content.

- Formalized signatures: Similarly, a format for signature files is provided. Here, signature declarations for mathematical symbols are attached to the CDs the symbols are defined in, by using file naming conventions. Since signatures depend on the type theory used to express them with, it is possible to provide more than one signature file per CD, and to distinguish those signature files that correspond to the same underlying type theory. Two such type systems were provided as non-normative extensions to OpenMath 1.0:

  - Small Type System: distinguishes constants, "binders," regular operators, "nary" operators, and "nassoc" operators; complex type "mapping" corresponds to OpenMath Application Objects.

[19]Note that OpenMath 1.0 was *not* defined as an XML application, however.
[20]This format goes back to the second OpenMath prototype definition by A. Diaz and G. Gonnet, 1996.

  - Strong OpenMath: based on the Extended Calculus of Constructions (ECC), this provides a detailed type system for OpenMath Objects with explicitly typed variables; complex types are provided that correspond to Application and Binding Objects.

- Web library: All definition files are accessible from a well-known website using a simple file naming scheme.

Ctadel was written in Prolog; we used a SWI-Prolog implementation as a base. No formal handbook of the Ctadel language existed, but its author (R. van Engelen) was very helpful, and the Ctadel source code was available, too.

With just a few exceptions,[21] we implemented the full glory of OpenMath 1.0 in Ctadel, including all of the MathML Content Dictionary Group. Because of the additional complexity of the new OpenMath standard, and because of the larger size and complexity of the set of symbols we had to consider, the task of incorporating OpenMath capability into Ctadel was more time consuming than previous projects had been. On the other hand, Ctadel's exceptionally clean design greatly facilitated the process of defining the OpenMath "Phrase Book", i.e. the translation between external OpenMath and internal Ctadel representations of a complex mathematical expression.

5.4.1 Parsing and Generating OpenMath in Ctadel

Since all OpenMath file definitions were accompanied by a formal XML DTD ("Document Type Definition"), it was straightforward to parse or generate any of the OpenMath formatted files using an off-the-shelf XML library for Prolog.[22] The intermediate result in both directions was a Prolog term which represented the syntactic structure of the XML text, i.e. it used the concepts of XML ("entity", "element") as heads of the term.

These terms were transformed into (or from) a more Prolog-like term structure with the XML elements' tag names as heads and the tag attribute list and the list of components of the elements as two arguments. In the case of OpenMath Objects, the set of possible tag

[21] We did not implement hexadecimal floating point representations, nor did we implement signatures, nor the byte code encoding.

[22] In practice, however, the library we used needed to be fixed in order to process these file formats correctly.

names was small – the tag names used to denote the different kinds of basic (e.g. OMS, OMV, or OMI for symbols, variables, or integers) or compound (e.g. OMA for application) OpenMath Objects. This corresponds to the data structure level representation that we had advocated a few years earlier [1].

These representations of OpenMath Objects at the abstract data-structure level of abstraction were transformed into (or from) a representation at the abstract mathematical structure level in Ctadel. Application objects were transformed into Prolog terms with the head of the Application object (in OpenMath terminology) as head of the Prolog term, provided this head was a symbol, for example, but Binding, Attribution, and Error objects required more specialized treatment. Symbols at this point still remained in the OpenMath namespace, giving a representation of the mathematical structure of the OpenMath Object with OpenMath symbols within Ctadel.

Finally, translation rules were implemented as Ctadel simplification rules that mapped between symbols and patterns of OpenMath and symbols and patterns of Ctadel. This phase of translation is known as the "Phrase Book" in OpenMath. The translation rules for OpenMath symbols were organized by their Content Dictionaries – one phrase book per content dictionary.

5.4.2 OpenMath Content Dictionaries and Content Dictionary Groups

Besides implementing import and export for files containing OpenMath Objects in the XML encoding, we also added code for interpreting OpenMath CD and CDGroup files.

The rationale for doing so was two-fold. First, it allowed a completely generic and extensible architecture for the Ctadel OpenMath implementation in that only the names of the CD Group(s) required by a specific Ctadel application would need to be specified, and the corresponding CDs and their corresponding Phrase Book code would automatically be loaded, including, most of all, the online documentation for OpenMath symbols that is part of their CD entries. Second, it allowed us to automate some of the tasks involved in reviewing the MathML CD Group, since CD entries typically included Formal Mathematical Properties (FMPs) in the form of OpenMath Objects which we could let Ctadel's simplifier

work on to help verify both the validity of the FMPs[23] and the validity of our OpenMath parser and Phrase Books.

Thus, the OpenMath compatibility code itself would not load any Phrase Books at all. Instead, a Ctadel command specifying which Content Groups were required would load the corresponding OpenMath XML files, extract the necessary information about member CDs, and load each such CD file along with its corresponding Phrase Book code.

Once all[24] the CDs and their Phrase Books were loaded, it was possible to initiate the simplification of the FMPs that came with the CDs in order to test both the CDs themselves and our implementation of their phrase books in Ctadel.

5.4.3 Lessons Learned

Unlike the Multi-CLP prototype, which was a full implementation of a real cooperative problem solver that utilized the knowledge communication capabilities of its OpenMath prototype implementations in an algorithmic manner, our Ctadel implementation of OpenMath 1.0 did not come with a ready-made set of tools to combine with it. Consequently, we did not bother (yet) to equip Ctadel with a low-level communication library like PVM, as we had before, but used simple file I/O instead. Since we already knew how to add PVM to Prolog if necessary, we felt that we could postpone this aspect until we actually wanted to start building the Problem Solving Environment we had envisioned in [5].

Prolog $\neq$ Prolog

Like some of the components of the Multi-Solver CLP system that we looked at in the previous section, Ctadel was implemented on top of a Prolog system. In many ways, therefore, the techniques we used were similar – for example, the use of a general-purpose off-the-shelf parser code driven by a syntax specification file as a first step in the parsing code, and the representation of OpenMath Objects as Prolog terms. However, there were also significant differences, since Ctadel was a small but full-fledged computer algebra system

[23]Several bugs in the FMPs were found this way, some of them rather obvious, leading us to suspect that we were the first to implement OpenMath CDs and CD Groups in a computer algebra system.

[24]It turned out that there was a most complex pattern of interdependence between the many Content Dictionaries making up the MathML CD Group with respect to the Formal Mathematical Properties in a CD and the external CDs they reference. It was therefore safer to wait until everything was loaded before attempting to simplify the first FMP.

implemented on top of Prolog, and therefore, for example, did provide the concept of a variable that OpenMath Variable Objects could map to, unlike a Prolog pure and simple did in our MultiCLP experience.

Cleaner computer algebra systems are easier

In our REDUCE implementation of the OpenMath prototype language we found that some concepts were exceedingly hard to translate properly, in particular the Boolean connectives which had a special meaning in REDUCE that did not work with the more general meaning that OpenMath (and our application) intended.

With OpenMath 1.0, an almost certain source of trouble with respect to implementation in a general-purpose CA system is the translation of an OpenMath Binding Object, unless the head of the Binding Object happens to be one that the CA system knows specifically how to handle. As we showed in [2], many CA systems are non-compositional because they do not have a systematic way of expressing (and handling) the concept of variable binding operators.

However, Ctadel was quite different in this respect, having a clean and general way to handle variable binding expressions. One reason for this probably is that its success in generating high-quality code from mathematical specifications of a numeric model crucially depended on making certain generalizations to the optimization technique of loop nesting reordering – generalizations which in turn crucially rested on abstracting out the notion of variable binding in a general-purpose, compositional manner, as opposed to the ad-hoc, special-purpose, operator-dependent manner in which the other CA systems usually handled it [144].

As an example, Ctadel understands any expression of the syntactic form

$$op(body, var_1 = range_1, \ldots, var_n = range_n)$$

to mean that *op* is an operator (or higher-order function) with a body *body* which is the scope of the local variables $var_i, i = 1 \ldots n$ that range over (or have the type associated with) their respective $range_i$ [144]. Compare this with a similar syntax in Maple that is, however, interpreted as special only by the specific operators within Maple that use the syntax rather than, as in Ctadel, systematically employed throughout the system.

In order to accommodate OpenMath 1.0 fully, however, and in order to handle all the test examples that came with it, it was nevertheless necessary to extend Ctadel's language by adding a λ-expression constructor, which can describe a function with local variables that are not bound to concrete values or ranges. This special constructor for anonymous functions is part of the core vocabulary of OpenMath, and there were cases in the Formal Mathematical Properties of OpenMath core content dictionaries that were not representable in Ctadel using only these existing language ingredients.

Content Dictionaries and Phrase Books

Needless to say, there were a few lessons we learned about writing Content Dictionaries (CDs) in the process of reviewing those we were given.

OpenMath's so-named MathML CD Group wants to provide those symbols that MathML 2.x Content Markup provides, with a compatible semantics. Since many of the same people were involved in the writing of both the MathML CD Group and MathML Content Markup specifications, this made a lot of sense.

However, there are a few cases where things are a bit tricky in a way that spills over into problems for Phrase Book writers. Here are a few examples.

Commutative vs. non-commutative multiplication

Take the example of the multiplication operator, which in MathML does not necessarily have to be commutative, but can be. For the OpenMath core CDs, the authors decided to split this into two distinct symbols – both having the same name, but living in different CDs, one being declared to always be commutative (using an Formal Mathematical Property (FMP) entry in its CD definition), the other being left unspecified in that respect.

In FMPs in this CD Group, on the other hand, the specifically commutative multiplication is not always used when its commutativity property is actually required in order for that FMP to hold. This is a sort of cognitive consistency problem, though it has been argued that technically, using the potentially non-commutative symbol is quite correct at least in those cases where commutativity could be inferred (e.g. by all arguments being numbers).

Consequently, certain test cases (i.e. certain FMPs in certain CDs in the MathML CD Group) did not simplify as expected – the translation had not succeeded in generating the cognitively intended, but only the technically and literally correct, equivalent of the symbol.

Levels of generality

A difficult choice when deciding on a particular mathematical concept to represent as a name in a CD is the level of generality in which it is to apply.

A case in point is that of the CD for the basic transcendental functions. Here, several authors published a paper with a new result showing that it is possible to provide a consistent set of definitions for the branch cuts of all the basic transcendent functions.

Forms of completeness

Although we did our best to provide a complete implementation of all the symbols in the MathML CD Group of OpenMath, the same cannot be said about the set of symbols that Ctadel had built in. In particular, as we pointed out later at the OpenMath Workshop in St. Andrews in Scotland, a CD of physical units was still missing,[25] as were the loop constructs that Ctadel was so good at handling.

Thus, an OpenMath implementation can aim for completeness in two different senses, namely completeness with respect to a particular set of Content Dictionaries, or completeness with respect to the capabilities of the CA system it is implemented for. Both have their own problems.

Implementing an OpenMath CD fully may mean the necessity of adding non-trivial functionality to the host system, such as a systematic treatment of variable binding in a system that does it in an ad-hoc way so far.

Implementing full access to the capabilities of a system, on the other hand, would almost certainly mean extending the current set of OpenMath names for mathematical concepts to include some that are not adequately covered yet but are particularly important to the system at hand. In the case of Ctadel, this included CDs for describing physical dimensions and units; in our first prototype MapleV–REDUCE connection, we stumbled upon the fact that MapleV sometimes outputs results in terms of a γ function that REDUCE knows nothing about (and which is not in the MathML CD Group, either).

Phrase Book writing

We once again learned as a lesson what we had, frankly speaking, started off with as a premise to the entire decade-long project, namely, that the ability of a symbolic

[25] Drafts of Content Dictionaries for physical units and measures now exist.

computing system to communicate knowledge with other systems needs to be implemented by people who know the system intimately at the deeper system levels of implementation. It was crucially important for the success of this particular project that the author of the Ctadel system was available to help with the implementation of numerous aspects of the OpenMath–Ctadel translation.

CHAPTER 6

THE LINGUISTICS APPROACH

In Chapter 5 we described how we discovered through practical experience in designing and building a system of cooperating heterogeneous computing systems that a knowledge communications language (and thus, a content markup language) decomposes into several layers and components in a natural way, and how we proposed that OpenMath in particular be structured in that way.

Indeed, we also noticed that the language architecture that we were finding to be of great practical usefulness for content communication, had an uncanny family resemblance to the architecture of human languages as it was being described by linguists working on formal grammars and formal semantics, and in 1995 we were the first to propose "to pursue the parallels between human language perception and mathematical information exchange to see how far it will carry" [6, 7] when designing content markup languages, though we regretted that we were only able to give "a very rough outline of the widely recognized components researchers have identified in human languages perception" and that it was "outside the scope of the paper to go into any more detail" than a couple of pages of describing the parallel between natural language architecture and our proposal for the structuring of OpenMath.

In this chapter, we will try to rectify that omission and provide a detailed survey of what the science of natural languages has discovered that may contribute to the proper design of content markup languages. The following chapters will then pick out a few key aspects of the linguistic theory surveyed here and apply them to specific content markup languages. Thus, the nature of this chapter is that of a survey motivating the core idea of this dissertation, namely, that the study of the structure of human language can teach us good designs for content markup languages, while the following chapters show just how exactly this can work for a few particularly interesting aspects of content markup language design.

6.1 Linguistics Ansatz

Within the research area outlined in chapter 2.1, our particular theoretical research interest is in investigating fundamental design principles for content markup and knowledge communication languages. The ansatz that we propose to follow is to analyze research results about the structure and underlying "design" principles of human language and communicationand to apply them to the design of content markup and knowledge communication languages, because so far, efforts at defining content markup languages have been producing flawed designs for lack of understanding of deeper issues (see Chapters 7 and 8 for several examples).

Content markup is an important current activity in computer science and its applications, as we saw in Chapter 4. However, as we will see in the following chapters, most content markup language designs to date have been flawed in subtle but important ways, perhaps because content markup language design has been treated more as an ad-hoc piece of art than as a science.

We believe that it is necessary to deepen our understanding of what it is that makes high-quality designs for content markup languages, and of what it is about them that would help or hinder exchange of knowledge encoded with them between a diversity of applications both on the Web and off, if we want to avoid in the future potentially costly errors like those we discovered in existing languages.

What is less clear is which road to travel towards this goal. In [1] (originally distributed as a working paper of the OpenMath community under the title of "OpenMath Objectives," and presented as a poster at ISSAC'95) we first proposed a promising approach to gaining an understanding of the encoding of semantic content in content markup languages. In that paper we proposed to analyze and use what has been discovered about the one "good" solution for content communication known to exist – human language and communication among human beings:

> "The decomposition of the task of transmitting mathematical information that we present [...] is "natural" in the sense that it reflects the structure of human language perception which "solves" a quite similar, though even more complex, task. [Here] we will take a short look at the structure of human language

> perception as it is modeled by researchers in theoretical and computational linguistics (in particular, syntax and semantics) and by mathematicians (working in the area of formal semantics) attempting to find mathematical models for important aspects of human language, and note the "mapping" between human language perception modules and the OpenMath model. [... O]ne may want to pursue the parallels between human language perception and mathematical information exchange to see how far it will carry." [1]

Fields of science that are currently studying communication in general and languages and their meaning in particular include philosophy, linguistics, mathematics, computer science, and cognitive science. Much has been discovered in recent decades both about the general structure and about the details of how human beings manage to convey their meaning to other human beings. A central discovery of the last half-century, generally attributed to Noam Chomsky, concerns what is now known as Universal Grammar – underlying grammatical principles that can be found in all human languages (including the sign languages spontaneously developing among the deaf [22]). Since, as for example Stephen Pinker [22] and Terrence Deacon [44] argue, the human language capacity has almost certainly evolved under great evolutionary pressure, the universals of human communication in general, and of human languages in particular, can reasonably be assumed to be high-quality solutions to fundamental problems of content communication in general.

However, the well-known fact that human languages are extremely hard to process on a computer despite their simple and elegant structure should also serve as a caution that some of the universal aspects of language are most likely due to the evolutionary accidents and biological constraints of the human species. Thus, while taking seriously the homology between people's ability to communicate their meanings to others and the goal of content markup to do the same among software systems is a very promising approach towards a deeper theoretical understanding of the latter, this approach does come with an inherent potential practical pitfall; indeed, through circumnavigation of this pitfall, the most important result of research that follows through this ansatz might well eventually be a better understanding of which language universals are due to underlying fundamental communication issues, and which to biological accidents of the human species.

Thus, the basic idea of our research is to evaluate the design principles that have been discovered in recent fruitful attempts at understanding the fundamental structure of human cognition in general and of human language in particular, and to extract from these design principles those that are due to fundamental issues in the communication of meaning, while leaving out those messy details that make human languages easier for humans to understand, but impossibly difficult for computers to handle.

Because the theoretical linguistics approach to content markup language design, presented in this way, sounds both obvious and impractical given the history of, say, machine translation research, it is important to realize that it was actually through repeatedly stumbling onto problems and concepts that looked familiar from linguistics while working on practical issues in content markup design and implementation during my involvement with the OpenMath effort (see Chapter 5) that it eventually dawned upon me that the linguistic approach also might be able to provide practical solutions to practical problems if considered from the right perspective. The fact that some of these solutions are not simple, and that logicians who studied the formal semantics of language learned quite a bit of unsuspected new mathematics in the process [25], is thus presumably due to the inherent complexity of the problem they attempt to solve.

6.2 Language Layers and Components

The human language facility has always been a phenomenon of deep interest to the scientist, the philosopher, and indeed to everyone else. The scientific literature on linguistic topics is huge, and has had a long, long history. There is no way that we can give full justice to it in the few paragraphs that we have available here to survey it, and we will therefore not even attempt to do so.

In his book *The Language Instinct* [22], S. Pinker frequently refers to different language phenomena as *engineering solutions* to problems that are inherent in the task of communicating meaning between speakers of a language. He surveys brilliantly the many universals that all known natural languages have in common, including those, like sign languages, with a completely different transport medium.

This includes in particular a rather strikingly universal fundamental architecture of layers and components of language. And as T. Deacon points out in his *The Symbolic Species*

[44], some of these components even appear to have their own specific substrates in the human brain, adding force to S. Pinker's characterization of these components as parts of the engineering solution of human language to the problem of communicating meaning.

Linguistic theory has had a long tradition of separating the study of human language into (at least) the following layers of language:

Phonemes are the basic ingredients of a language, comprising the set of sounds that a language uses to build, and distinguish words. Essentially all sound systems of natural languages around the world use phonemes that are distinguished along a small number of phonetic dimensions, but different languages use a different subset of these distinct sets of phonetic dimensions.

Morphemes are the basic building blocks of meaning in a language. Morphemes are typically combinations of phonemes, and are either words themselves (such as English *word* or *poet*) or particles (such as the English plural suffix *s* or gerund suffix *ing*) that need to be combined in meaningful ways with other morphemes to build words.

Words are then the basic building blocks of sentences. Simple words are built by combining one or more morphemes into a meaningful whole, and compound words are built from words. *Morphosyntax* is the set of rules that describes how words are built from morphemes and/or simpler words in a language.

Lexicon is the term used to describe the collection of words and their linguistically relevant properties known to speakers of a language.

Phrases are the complex structural ingredients of sentences of a language; a *sentence* is one particular kind of phrase, noun phrases and verb phrases are other universal ingredients in human languages.

> "One of the most intriguing discoveries of modern linguistics is that there appears to be a common anatomy in all phrases in all the world's languages." [22]

Phrases are said to belong to the same *category* if they are interchangeable: speakers of a language consistently either judge both or neither sentence grammatical if one phrase

is exchanged for another in sentences that contain either. Phrase boundaries can be recognized similarly by differentiating those sequences of words that act consistently in the interchangeability test from those that do not. *Syntax* is the set of rules of how grammatical phrase structures are formed in a language. The concept of a *Universal Grammar* recognizes that the phrase structures of human languages can all be described by a basic set of *principles* according to which they are constructed, and by a basic set of *parameters* that are set in specific ways in each language by choosing from a small set of possibilities.

Semantics is the term used to denote the rules that determine how a speaker of a language determines the meaning of a phrase of that language. Linguists distinguish between *lexical* semantics (that is the meanings of individual words, "recorded" in the lexicon), and *categorial semantics* (that is the rules that govern how the meanings of sub-phrases of a compound phrase combine to determine the meaning of the compound phrase). The semantics of human languages is widely held to be *compositional*, that is describable as an interpretation that mirrors exactly the phrase structure of an utterance.

Pragmatics is a term that is used to describe the aspects of meaning that depend on the context in which an utterance occurs, where the pure semantics of the utterance is thought of as its context-free meaning, so to speak. Context can be that of different parts of a narrative, but also cultural.

6.2.1 An Architecture for Content Markup Languages

A parallel architecture for content markup languages that is suggested by this layering of natural languages is roughly as follows:

Phonemes correspond to the characters used in an encoding of a content markup language. They are intimately tied to the *communication* layer, so to speak, and have little or no relevance at higher levels of a language.

Morphemes correspond to those ingredients of an encoding that a lexical analyzer would recognize: symbols and special syntactic markers such as parentheses.

Words correspond to those aspects of a content markup language that convey *lexical* meaning between communication partners. A *morphosyntax* describes how words are formed from simpler meaningful components.

Lexicon corresponds to some kind of knowledge base of words and their meanings.

Phrases are built from lexical entries and special syntactic markers according to a set of recursive *syntax rules*. Although the specific lexicon and morphosyntax may vary with specific encoding needs, the phrase structure should be essentially universal.

Semantics consists of two distinct aspects, namely the universal *categorial* semantics of the universal phrase structure on the one hand, and the language-specific *lexical* semantics of the words of a language which evolves fairly rapidly as the language assimilates new vocabulary or looses disused concepts. The interpretation of a phrase in the language should be *compositional*, that is, mirror its syntactic structure faithfully.

Pragmatics adds to this a knowledge about what to do about a piece of content markup once it has been received, and how to extract that knowledge both from the utterance itself, from those that preceded it or were made by the recipient itself, and from the general and specific "cultural" world knowledge common to the interacting software agents.

Compare this with the architecture that we proposed for OpenMath in [7, 1] in Figure 6.1.

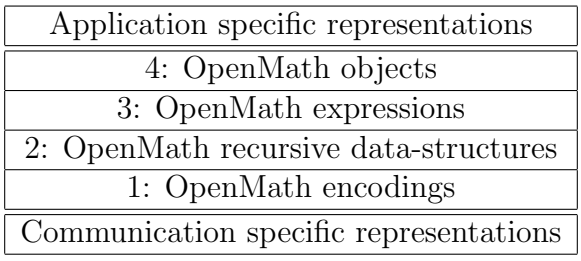

Figure 6.1. Proposed Layers of OpenMath

The idea here was that layer 2 corresponds to a syntactical analysis, layer 3 to a categorial semantics, and layer 4 to a fully semantically analyzed OpenMath Object. Layer 4 thus adds the lexical semantics to layer 3 (the lexicon, as we termed it then, does not appear in this view of the architecture, as it is the intermediary between layers 3 and 4 and hence implicit in the architecture, as can be seen in Figure 5.4).

We proposed that both the syntactic structure and the categorial semantics of OpenMath Objects be considered as basically constant and universal across OpenMath implementations, while for each OpenMath application, both the specific encodings it supports and the specific lexical entries it recognizes may be a parameter taken from a finite set of possibilities. It was also thought that the application itself, in the process of interpreting an OpenMath Object, would apply its own background knowledge to interpret the object pragmatically.

The proposed architecture therefore mirrored quite well that of the architecture of language usually ascribed to it in linguistics.

6.2.2 The Structure of OpenMath 1.0

The authors of the first official version of OpenMath [10], published a few years later than the above-cited paper, did not follow our recommendations to the letter. Instead of the four layers distinguished in [7, 1], only two layers are recognized in [10] – namely, the OpenMath Encoding and the OpenMath Object layer. In addition, the concept referred to as a "lexicon" above is renamed to "content dictionary" in OpenMath 1.0.[1]

Although only two layers are explicitly stated in [10], the actual architecture proposed in that document, when analyzed in the terms of the above sections, yields three layers, with the central two layers of our recommendation (OpenMath recursive data-structure and OpenMath expression) replaced by a single layer (named OpenMath Object, a term our recommendation used for the fourth layer that adds information from the "lexicon").

Consequently, the semantics of OpenMath 1.0 "Objects" are under-specified, as Chapter 8 shows. The concept of a "categorial semantics" (or skeleton semantics) that is introduced in that chapter corresponds to the "conversion of objects of level 2 to level 3" in our recommended architecture; by leaving out the distinction between abstract data structures

[1] Indeed, it went through several name changes during the first few years of discussion on OpenMath, including "lexicon," "dictionary," "context dictionary," and "content dictionary."

and their semantic interpretation, OpenMath 1.0 paid insufficient attention to the semantics, as serious design errors that we recently discovered (and report on in appendix 8) show.

Similarly, heated discussions have taken place within the OpenMath community on the inconsistency in OpenMath with respect to the treatment of floating point numbers as pure IEEE floating point data formats on the one hand and that of integers as arbitrarily large mathematical integers regardless of their underlying data structure representation on the other. In our model, the former would clearly have belonged on the OpenMath recursive data-structure layer, and the latter, on the OpenMath expression layer. In the same vein, while semantic level labels-and-references in the form of variables and bindings have been included in OpenMath 1.0, structure-sharing mechanisms that we recommended at the data-structure layer below were left out of the standard – and thus come up time and again in OpenMath meetings as things to add.[2]

Thus, "simplifying" the structure that we originally proposed as a parallel to the structure of human language has turned out to be quite problematic.

6.3 Morphological Structure

In natural languages, the morphosyntactic structure tends to be significantly simpler than the syntactic structure of the language. There are two main uses for structured words, both found in essentially all languages of the world. First, grammatical markers help distinguish the grammatical categories of words in many languages, including the role of a required argument. Second, compounding of words is a way of making new words out of simpler existing words; in this case, due to the simple structure of morphosyntax, we usually have a main word modified by a second word, with an "is-a" relationship holding between the compound word and its head: an icebox *is a* kind of box.

Morphosyntax is productive – it can be used to make new words never before seen or heard – and it is recursive – it can be used to make arbitrarily long words. German is particularly famous for this, but it is trivially true in any language that assigns words to numbers.

[2]OpenMath Workshops in Yorktown Heights (1997), Eindhoven (1999), Nice (2002) and Pisa (2002) are some of those that had this topic on the agenda.

In content markup languages, we have particular use for a morphosyntax for the purpose of compounding words, although we have been on record as recommending one for the purpose of assigning words to categories, too, and the Maple CA system user language uses a simple morphosyntactic feature to distinguish between "noun" and "verb" versions of the same symbol, e.g. the capitalized `Int` standing for the noun *integral* and the lowercase `int` standing for the verb *integrate*.[3]

In [21] we mentioned that the concept of a simple morphosyntax that provides for compound word construction in human language, when transferred to content communication languages, translates into a proposal to make explicit a simple morphosyntax for names. As an example, the concept of "proper subset" ("$\texttt{prsubset}_{\texttt{sets1}}$" in OpenMath and `</prsubset>` in MathML) contains the concepts of "proper" and "subset", both of which appear in the names of many related mathematical symbols; in a "good" content communication language, this would be made explicit and thus available to automatic processing by, e.g., a renderer which could take the subset symbol and superimpose on it the not-equal glyph that corresponds to the concept of "proper." We therefore advised against naming conventions that allowed a name like "prsubset" instead of "proper-sub-set."

As another example, in [64] we proposed a "name-space of name-spaces" to facilitate nesting of name-spaces; a simple morpho-syntax like the one for directory paths would serve this purpose more easily – we might have a symbol $\texttt{measure}_{\texttt{functional}_{\texttt{analysis}}}$ in the namespace $\texttt{functional}_{\texttt{analysis}}$ which in turn would be in the namespace `analysis`, or we could distinguish between several meanings of functions for the real, complex, formal power series, or generic readings respectively of the sine function by introducing appropriately compound symbols for the sine-function, the real-sine-function, the complex-sine-function, and so on.

In OpenMath, the equivalent of "words" would be the "basic" OpenMath objects and the several constructors. Each of the basic objects belongs to an explicit subclass of the class of Basic OpenMath objects – in OpenMath, words are structured, albeit with a very simple and fixed structure. Names have a simple structure in OpenMath, as they belong to Content Dictionaries whose names are part of an OpenMath Symbol. However, there is no

[3]The semantic difference between verbs and nouns in Maple is that verbs are simplified (in our example, the symbolic integration is performed if possible), but nouns are "inert" and only displayed.

nesting in OpenMath's morphosyntax, which makes for somewhat arbitrary limitations in Content Dictionary design.

In MathML, "words" correspond to the XML tags, XML entities, and XML CDATA strings. The latter two are atomic and unstructured "morphologically," but the first of these has a simple but extensible structure based on the ability to add arbitrary attribute names and value strings to a tag. However, there is again no nesting in MathML compounds, thus providing a limited amount of usefulness of word composition here, too.

This aspect of the linguistics parallel for content markup language design is being mentioned here for illustrative and motivational purposes only, however. We will leave the working-out of principles of a morphosyntax for content markup for future investigations.

6.4 Syntactic Structure

This section is based on Stephen Pinker's excellent 1994 book "The Language Instinct" [22] because it is the most accessible introduction and overview on the beautifully simple syntactic structure of human language that I am aware of,[4] and in particular its core chapter titled quite immodestly "How Language Works" is highly recommended as providing detailed and accessible explanations of the aspects of the universal grammar of language that we discuss here.

> "One of the most intriguing discoveries of modern linguistics is that there appears to be a common anatomy in all phrases in all the world's languages. [22]

As we said earlier, this is an indication that this "common anatomy" is the visible "engineering solution" to an underlying set of fundamental problems of knowledge communication. It is therefore worthwhile to take a closer look at that universal architecture of language.

According to Pinker, the first principle of phrase structure is that every phrase has a *head* that determines the meaning of the phrase.

All content markup languages that I am aware of concur.

Secondly, phrases are made up of their head, its *arguments*, and its *modifiers* or *adjuncts*, each of which plays a designated role in the meaning of the phrase. The head and its

[4]However, the idea of an homology between human language syntax principles and content markup language syntax principles is based on classes I took at the University of Cologne in the mid-to-late 1980s.

arguments make up a discernible sub-phrase, which together with the modifiers then forms a full phrase. The difference between arguments and modifiers is that the former are mandatory and the latter optional; an "argument provides information that is inherent to the meaning of the phrase, the adjunct gives just a bit of additional information to help identify what we are talking about."

Pinker goes on to summarize the fundamental principle of Chomsky's X-bar theory of syntax as follows (where X stands for any of the categories of :

> "A phrase $[(\bar{\bar{X}})]$ consists of an optional subject, followed by an X-bar $[(\bar{X})]$, followed by any number of modifiers.
>
> An X-bar consists of a head word [determining what kind of phrase it is], followed by any number of role-players." [22]

Thus, the syntactic structure of a content markup language suggested here is a tree, the internal nodes of which all have this structure (in prefix form):

$$\bar{\bar{X}} \rightarrow ([subj]\ (X\ arg_1 \ldots arg_m)\ adj_1 \ldots adj_n)$$

Consequently, on my suggestion, OpenMath 1.0 uses the terms "head" and "arguments" when referring to the corresponding components of OpenMath Application and Binding objects.

There is also an intriguing possible correspondence between the components of an OpenMath Attribution object and modifiers ('adjuncts'); however, there are currently some important semantic distinctions between the intent of OpenMath attributes on the one hand and the role that adjuncts play in language.

According to our ansatz for finding good design principles for content markup languages such as OpenMath by analyzing natural language homologues, this means, on the one hand, that including Attribution objects is probably good syntax, but on the other hand, that the semantic rationale behind Attribution objects might bear reconsidering. Interestingly, there have been quite a few discussions on uses of Attribution objects for purposes that correspond more closely to that of 'adjuncts' described in the quote above.[5]

[5]E.g. at the OpenMath workshops in Eindhoven (1999) and in Nice (2002).

A common pattern of a syntactic proposal for representing mathematics on the web is therefore a tree, every internal node of which would consist of an operator (the head), its arguments (which are inherent to the semantics of the expression thus represented, usually positional), and attributes (which are incidental information; safe to ignore if only 'meaning' is to be interpreted; optional, or named, arguments). The 'X' – or syntactic category – is treated differently in different proposals. In some, all operators are treated alike (i.e. there is only a single category). In others, an operator may have an arbitrary, consistent number of categories associated with it. In some, there are different categories of brackets instead (function application, quantification, relation, etc.). Often, the category is implicitly a part of the lexical entry defining the head, and doesn't appear explicitly in the structure except perhaps through a separation of name spaces.

MathML-Content even has something that might be taken to be similar to the subject role in the above structure, namely the declaration element that is an optional first element of any MathML-Content element which is not counted as an argument.

6.4.1 Agreement

Another "engineering solution" common to all languages is that of *agreement* which allows a phenomenon called *scrambling*, i.e. rearranging arguments not only in a single predefined positional order but in several different orders. This corresponds to a common technique in programming languages that allow one to use "named arguments," with "naming" of arguments implemented, for example, by case marking in natural languages.

This appears to be a useful suggestion for content markup languages, and indeed MathML's qualifiers appear to be playing a comparable role in that language. Indeed, these qualifiers appear to be treated in that language as "bits of crystallized grammar", as Pinker puts it in reference to a peculiar phenomenon of human languages.

The distinction between content words (an open and constantly changing set of words whose meaning is determined by a lexical semantics) and function words (a "closed club that resists new members" [22] of words that are meaningless except as syntactic markers) is important if we want to study the engineering solutions arrived at by the world's human languages, and to draw useful lessons from these studies.

The interesting question then becomes: what is the general structure of the syntax of human languages in this broader sense of the word? What are the phenomena that are consistently treated at this syntactic layer of human language (rather than, say, through content words)? Why might this be so? Is that reason transferable to the case of knowledge exchange between computers? What is the range of choices different languages have made to 'implement' such universal syntactic phenomena (e.g., word order vs. case marking; function words vs. affixes). What is the corresponding range of choices in the engineering task we are considering? Which are the *pros* and *cons* of such choices given the different nature of humans and computers?

In the recent development of OpenMath, precisely this kind of a point was raised during a discussion on how to represent the fundamental concept of variable binding in that language: should one introduce a new syntactic constructor, or should one instead introduce a special symbol, lambda, for this purpose? We had mentioned the latter as a possible resolution of a problem in the version of OpenMath that preceded that discussion [2], but in this particular case the OpenMath group decided to go for a new syntactic constructor, the OpenMath binding object, instead.

6.4.2 Topicalization via Deep Structure and Surface Structure

Another engineering solution Pinker discusses is that of the difference between deep structure and surface structure of phrases in human language, which is intimately related with the existence of inaudible but provably present "traces" in the syntactic surface phrase structure. These traces are references to sub-phrases that were moved away from their syntactic "home" position into a more marked position in the overall phrase structure in order to "topicalize" those components of the utterance.

This suggests that 'topicalization' may be an important ingredient in a semantic markup, too. As in human languages, topicalization is more than likely to be implemented by moving the full representation of the topic – the 'important' (and most likely recurrent) part of the object – to the 'front' and 'top' of the tree structure, and to refer back to it in those places where it eventually will turn up, again and again, in different roles and different contexts deep within the object. Similarly, topicalization may be useful in subexpressions that 'happen' to have their own sub-topic.

OpenMath 1.0 does not provide a syntactic mechanism for topicalization in this sense, but MathML does in the restricted form of "declare" elements which may be included at the top-level of a MathML "math" element.

6.4.3 Traces: Labels and References

Labels and references are important ingredients of a semantic markup language; our understanding of the engineering solution arrived at in human languages suggests that the syntax should provide at least one kind of mechanism for this – the semantics component may offer more.

At this point, both MathML and OpenMath are restricted to two kinds of "labels" in the above sense, namely variables (which may be bound or free in subexpressions) and symbols (which introduce global names for concepts). As has been mentioned before, the addition of labels and references at the data-structure level for sub-structure sharing (which is known to have a potentially exponential space savings effect) has been proposed frequently for OpenMath (see for example [7]) and presumably also for MathML.

6.4.4 Scope and Binding

Scoping rules are also an important ingredient in human language that respect the syntactic tree structure of a sentence. By our ansatz it follows that content markup, too, should provide a mechanism for representing the notion of scope, both for operators and variables, and both locally and globally.

It turns out that there are several kinds of binding 'implemented' in human languages (PRO, pronouns, etc.) with different binding scopes (sub-phrase – traces and reflexives; conversation – pronouns; culture – names). For example, one of the "parameters" that describe a particular language's structure is the set of phrase boundaries (NP, VP, IP) that act as barriers across which trace bindings (and thus, movements) are forbidden by that particular language. As an example, Norwegian has a smaller set of these barriers than does English, meaning that it allows "longer-distance movements" than English.[6] In other words, there are mechanisms in place that determine the scope of trace bindings.

[6] J.T. Lønning, personal communication, 1988.

In addition to the 'label-reference' mechanism described above, another basic mechanism is 'co-indexing,' or the notion that parts-of-speech refer to the same thing. Traces we have met, and their scopes are restricted to certain sub-phrases. Reflexive pronouns (*himself, herself*), too, are bound in this way within a phrase, and again, different languages have different rules as to which parts-of-speech 'bind' them, i.e. define their scope.

'Regular' pronouns (*he, she*), by contrast, may refer to sub-phrases outside the scope of a single sentence – essentially, they may be considered free variables of a sentence. They only make sense, usually, in an ongoing conversation.

Names, finally, can retain their meanings in a group across conversations.

6.5 Semantic Structure

This section is based on "The Handbook of Logic and Language" [25] (1997) which features very similar authors and topics to those of the 1988 "Workshop on Computational Linguistics and Formal Semantics" at Istituto Dalle Molle in Lugano, Switzerland, where I first became acquainted with research into the formal structure of the semantics of natural languages.

Two chapters of the Handbook of Logic and Language provide advanced theoretical background for two crucial aspects of my research. The compositionality principle that is the topic of Chapter 7 is discussed at length in T. Janssen's chapter "Compositionality" [23], with additional insight provided by B. Partee's chapter [145] on Montague Grammar. The concept of a categorial grammar that I explore in Chapter 8 is discussed in depth and at length in M. Moortgat's chapter "Categorial Type Logics" [24].

6.5.1 The Compositionality Principle

A key case in point of our linguistics approach and of its usefulness for the proper design of content markup languages, namely the compositionality principle, will be discussed in great detail in Chapter 7. Through our personal effort, it has proved quite influential in the designs of both MathML and OpenMath.

The compositionality principle (*the meaning of a compound expression is a function of the meanings of its parts and the syntactic rule that glues the parts together*) is an important

guideline in current research into the detailed logical structure of the meaning carried by human language [23]. One of the results of this principle with respect to the structure of human language has been discovered to be the need for special syntactic constructs to handle binding; in the case of mathematical formulas, we argue for this in [2, 3].

When looking for good ways to represent mathematical formulas, it is thus the treatment of variable binding operators like integrals or quantifiers that is a particularly interesting test case where the application of the compositionality principle proves especially useful [2, 3].

In [2] we have argued that compositionality of a content communication language is necessary for scalability and correctness. Examples of languages that represent "content" mathematics using non-compositional constructs are given in that paper and in Chapter 7, and Chapter 7 analyzes a few constructs from one of these languages in considerable detail. In [2] we show how non-compositionality can severely limit the scalability of a content markup language design, and in Chapter 7 we show how correctness can suffer.

6.5.2 Categorial Type System

In [2] we noted a particularly important consequence of the compositionality of a content communication language:

> "Semantic information processing on a computer always means applying a semantic interpretation procedure to some concrete tree- or graph-like representation of an object that is being interpreted, and to then represent the new interpretation given to the object thus treated in another concrete tree or graph.
>
> For a language for expressing semantics for exchange among computer programs, this means that it must perforce define a basic interpretation procedure – how to interpret the parts, and how to interpret the combinations of parts. Compositionality is important because it means that this interpretation algorithm can be defined completely with respect to how to interpret combinations of parts once you have interpreted the parts.
>
> Defining a skeleton interpretation involves defining the structure of the lexicon and that of its entries, but no specific lexicon entries should need to be referred to in its definition. Thus, compositionality means that one is able to provide a

> firmly grounded scaffolding from which one can work to extend the scope of your language by adding only lexical entries, while keeping the scaffolding intact." [2]

In the field of Formal Semantics within the research area of Linguistics, this concept of a "skeleton interpretation" is perhaps best known under the term "categorial semantics." In this section, we introduce this concept for content markup or knowledge communication languages, and apply one particular aspect of this concept, Categorial Types, to OpenMath 1.0 in particular.

Chapter 8 goes into a lot more detail in applying this concept to OpenMath, and makes concrete suggestions for fixing some of the problems that the research it reports on uncovered in OpenMath 1.0.

Formal Semantics for Content Markup

The specification of a semantics for content markup languages like MathML-Content and OpenMath appears to be a thorny issue. Both languages have essentially side-stepped the issue by declaring their semantics "informal" and thus, open to interpretation.

In the case of OpenMath, two type systems have been published alongside the OpenMath Standard document in order to provide a more (ECC [32]) or less (STS [31]) detailed semantics for OpenMath Objects. Thus, the OpenMath stance appears to be that many type theories exist, and since any formal semantics induces such a type theory, any number of such semantics for OpenMath must be allowed to co-exist peacefully.

I am not aware of any similar project for MathML Content markup; the MathML philosophy appears to be that its semantics is informal by definition and no formal semantics is desired (or possible?).

On the other hand, developers of another knowledge communication language, the Knowledge Interchange Format (KIF [93, 94]), invested considerable energy in providing a fully formal semantics (but not type theory) for their entire language. Ontologies based on KIF have included "engineering mathematics," making KIF directly comparable to MathML and OpenMath despite its quite different designated application area of intelligent agent communication.

While we are interested in providing a categorial semantics for content markup languages, we will here restrict our attention to the simpler case of a categorial type system.

Completeness of Formal Semantics vs. Extensibility of Content Markup

The main reason for avoiding the specification of a formal semantics for a content markup language, for example in the form of a formal type system, appears to be grounded in the conflicting demands, on the one hand, for completeness (required by a formal type system) and, on the other hand, for incompleteness (required for extensibility of a content markup language to the marking-up of novel concepts).

Another reason appears to lie in the fact that different type theories make different choices when it comes to weighing type inference power (and thus the set of constructs that are recognized as valid) against automatic validation speeds (or indeed decidability). Defining too weak a formal type theory, it can be argued, would run the very real danger of prohibiting the use of even quite common scientific formalisms, but would allow one to require its implementation by "conforming" applications. Defining too strong a formal type theory, on the other hand, would allow one to formalize a wide range of conceptual structures, but the formal proof theory for such a type system might be prohibitively expensive to implement – if a decision procedure should exist at all.

Categorial Semantics and Types

The concept of a categorial semantics in general, and of a categorial type system in particular, tries to provide a good compromise between the two extremes of both these dimensions.

A categorial semantics is *complete* with respect to the syntactic structures provided by a content markup language, but it is at the same time *extensible* (or rather, agnostic) with respect to the set of atomic "symbols" in the language. By restricting itself to the structural semantics of a content markup language and leaving the lexical semantics as "someone else's problem," a categorial semantics can be kept simple enough to be implemented easily even if its actual implementation may not actually be required by a standard. By providing a clean conceptual interface to "external" type theories on (a subset of) the atoms of the language, it acknowledges the fact that different formal semantics are possible and, indeed, useful, but by providing a single common structural semantics for the language it makes sure that any two such semantics agree at least on the basic meanings of an expression.

Origins of Categorial Semantics

We have "stolen" the concept of a Categorial Semantics from the field of Formal Semantics, which in turn belongs to the field of Formal Linguistics. In Formal Semantics, sometimes also known as Montague Semantics, it is a fundamental tool for studying the structure of the semantics of language. More specifically, "categorial type logics" (thus the title of a chapter [24] in a recent survey of the field, the "Handbook of Logic and Language" [25]) have been studied extensively in this field.

We will therefore adapt a formalism from this field, known as Lambek calculus, for the purpose of illustrating its usefulness in the much more recent field of content markup languages.

Compositionality and Lexicalism

A categorial semantics for a language can only exist if it obeys the twin principles of *Compositionality* and *Lexicalism*. A strong version of the compositionality principle can be expressed mathematically as the requirement that the semantics be a homomorphism between the syntactic and semantic algebras of the language. Lexicalism, in addition, requires all atoms of the language to be assigned a type or semantics exclusively via a "lexicon," leaving the semantics of compound expressions of a compositional language up to the homomorphic images of the language's syntactic constructors.

Note that this also requires a separation of the set of syntax rules of a language into two distinct types – those that produce "atomic" expressions from, say, bits and bytes, and those that produce compound expressions from atomic ones.

6.6 Pragmatics and Beyond

So far, work on content markup languages has focused on syntax and semantics issues, and little work has been done on higher language layers. The reason for this is not so much that people have been unaware of the distinction – indeed, the naming of "commands" (e.g. *integrate*) as opposed to "operators" (e.g. *integral*) has been a topic of discussion from the start of the OpenMath project. This is not surprising since the Maple Computer Algebra system, whose developers were initially among the primary movers of that project, makes the distinction between "nouns" and "verbs" quite explicit. Rather, I suspect that it is the

need to get the semantics level, the current focus of the OpenMath project, right before higher levels can be attacked.

Consequently, projects like OMdoc [90] that might be classified as working at this level of content markup languages explicitly take OpenMath as given, and go from there.

In this dissertation, I will not go into the design principles of this layer of a content markup language at all. The main reason for this decision is that the scientific discussion on this aspect of human language has not reached the degree of fruition that the study of the syntax and semantics of human language have in recent years. Hence, the linguistics approach that I am advocating can not be as fruitful.

Nevertheless, I would like to point out one particular linguistic design principle that I have identified at this level.

Notice how the Maple system makes the semantics/pragmatics distinction by directly referring to the linguistic distinction between "nouns" and "verbs". Indeed it does appear that human language makes some kind of language layer role distinction that is tied to the distinction between the noun phrase ("NP"), verb phrase ("VP"), and "IP" syntactic categories of a sentence. Taking this seriously, we can make a couple of interesting observations:

- Languages have systematic grammatical means of transforming a verb to a noun and vice versa (*integrate* vs. *integral*). Thus, once content markup languages have defined concepts in the form of "nouns" (on the semantic level), they should be available in a systematic way in a content manipulation language that they are embedded in (on the pragmatics level).

- Languages systematically allow the mutual embedding of noun phrases and verb phrases. In current proposals, "noun phrases" (semantic markup) may be embedded in "verb phrases" (pragmatic markup), but not vice versa.

CHAPTER 7

THE COMPOSITIONALITY PRINCIPLE

7.1 Introduction

In its most commonly cited form, the compositionality principle states that

> "the meaning of a compound expression is a function of the meaning of its parts and the syntactic rule by which they are combined." *(B. Partee et al. quoted in [23])*

Thus, a language is called *compositional* if it has a syntactic structure and a semantics that together obey this principle.

Mathematically, the compositionality principle is generally modeled in the literature as the somewhat stronger requirement of the existence of an interpretation that acts as a homomorphism from a syntactic algebra to a semantic algebra.

7.1.1 Compositionality of Natural Languages

According to the survey of *Compositionality* [23] in the *Handbook of Logic and Language* [25], the *compositionality principle* has been a powerful research guideline in those fields of the linguistics of natural languages that attempt to understand how human languages serve to communicate meaning among individuals. The field of Formal Semantics, in particular, has been gaining profound understanding of this question from studying the ramifications of this principle.

The principle has been a successful research guideline for linguists in the form of the hypothesis that all natural languages are compositional, motivating the search for compositional analyses of the full range of linguistic phenomena as well as for language constructs that do not easily admit of such an analysis. The debate about whether or not (or

rather, to what degree) natural languages are compositional continues to rage in the literature (the first issue of the 2001 volume of the Journal of Language, Logic, and Information [146] was devoted entirely to this debate, for example), but the consensus appears to be that it is very fruitful to assume that they essentially do obey the principle.

7.1.2 Compositionality of Computer Languages

Given the above definition of compositionality, it only makes sense to ask if a language obeys the compositionality principle if it professes to transmit *meaning*. In the special case of computer languages for mathematics, this rules out judging presentation languages like LaTeX, Presentation-MathML or any other language for representing the two-dimensional layout of formulas. For such systems, the question is moot.[1]

Knowledge communication languages such as the Knowledge Interchange Format (KIF), OpenMath and MathML-Content, as well as the user-level languages used for interacting with Computer Algebra systems or theorem provers, however, are among those that do claim to represent the *meaning* of the mathematical formulas they express, and these languages therefore may reasonably be investigated as to, and judged by, the degree to which they respect or violate this principle [2].

We will see in this chapter that many Computer Algebra systems' user languages violate the principle of compositionality in their representation of *generalized quantifiers*. In practice, reduced scalability and extensibility for such systems means that adding new operators that show properties of generalized quantifiers requires much more work than adding new simple functions. However, because such systems possess an operational semantics, it is usually possible in these systems to compensate for the problematic representation by adding coding overhead.

[1] Note, however, that by replacing the term *meaning* by *formal semantics*, the definition of compositionality is easily extended to general formal languages and their syntax and semantics. In this somewhat different sense, the compositionality principle has played an important role in several sub-disciplines of computer science, as [23] points out. Denotational semantics in particular implicitly assume a compositional approach to formal languages.

7.1.3 Compositionality of Content Markup Languages

Purely declarative content communication languages like KIF, OpenMath or MathML-Content, however, are intrinsically devoid of such compensation mechanisms. Since OpenMath in particular has extensibility as a core requirement [26, 7, 1], obeying the compositionality principle becomes a matter of vital concern, where in the Computer Algebra systems that it was originally designed to help communicate with each other, failure to obey the principle may "only" have been a nuisance.

Thus, the compositionality principle originally entered the MathML-Content and the OpenMath discussion through our noting that this guiding principle of linguistics was violated in common representations for certain classes of mathematical formulas in classical computer algebra and knowledge communication languages, including older draft versions of OpenMath and MathML-Content. It has since fully proved its value as a design guideline for these two content markup languages.

We have also studied how both correctness and scalability suffer in two versions of the Knowledge Interchange Format (KIF) when the compositionality principle is violated. In particular, in section 7.3 we report our finding that KIF 3.0 contains at least two fundamental constructs that violate the compositionality principle, and we will show how scalability and correctness suffer from this. For dpANS-KIF, on the other hand, we will show in section 7.4 how "fixing" such problems by leaving out a whole class of operators has a dramatic effect on the expressiveness of the language, and that none of the non-compositional mechanisms made available in the language are capable of circumventing this problem.

These results vindicate a controversial proposition we first made in [26, 1], and which forms the core of this dissertation. There, we proposed to pursue a line of research into improved designs for OpenMath that would take into account the structural parallels between languages for communicating *meaning* between computer systems on the one hand (e.g. OpenMath) and languages for communicating *meaning* between people (human languages) on the other. A followup article [2] used the compositionality principle as a case in point for a refinement of this original proposal.

7.1.4 Generalized Quantification and Compositionality

Formal languages for content communication usually follow a basic design that is very similar to the LISP style of representing symbolic expressions, utilizing a tree-like prefix syntax structure, and interpreting a node as application of the first child or *head* of the node to the rest of the children or *arguments*.

This basic design of representations for the applicative structure of symbolic expressions is clearly compositional in nature, and since it is almost universally employed anyway, claiming that it is an example of a compositional representation does not help language designers change their representations for the better.

It is thus when content markup languages need to represent concepts that go qualitatively beyond the basic pattern of functional application that we have to look for a real impact of the compositionality principle as a design principle for content markup languages.

In order to understand what it means for a concept to go *qualitatively beyond* the basic pattern, let us recall the definition of compositionality of a language which says that the meaning of a syntactically compound expression of the language must be a function of the meaning of its parts and of the syntactic rule that governs its composition.

For the designer of a language who aims for compositionality, this means that qualitatively different concepts are those that require qualitatively different interpretation mechanisms and that *therefore* require distinctive syntactic rules to govern their construction in their respective representations within the language. After all, if two such constructs were indistinguishable syntactically, they would have to be subject to the identically same interpretation procedure rather than the qualitatively different ones we assumed they required.

If, in addition, we require more strictly that the compositional "meaning" function be computable, the compositionality principle yields the design recommendation that any semantic interpretation pattern that occurs in a potentially unbounded number of contexts in the language, and thus especially in contexts that serve to meet an extensibility requirement, needs to be represented with its own dedicated syntax rule.

In formal languages from computer science or mathematics, a common aspect that is well established as being qualitatively different from the basic ingredient of functional application is that of *abstraction*, or more generally, that of variable binding and scoping.

Application of the compositionality principle to the design of content markup languages therefore yields as a design recommendation that abstraction or binding be represented using special syntactic constructors for this purpose. In the stricter form it also yields the recommendation that such a syntactic constructor be *open*, i.e. usable with regular extensions to the language rather than limited to a small number of fixed contexts.

Thus, we will analyze in the following the representation of *generalized quantifiers* (i.e. of operators that bind variables, including regular quantifiers, lambda abstraction, series, sums, products, integrals, and many more) in several versions of several different content markup or knowledge communication language specifications.

Later, in section 8.7.9, we will see that the same line of reasoning can be used to identify yet more language ingredients for a content markup language that require special syntax, but are not always given one.

7.2 Compositional Analysis of a Mathematical Formula

In the next section we will consider the different ways that definite integrals are represented in mathematical formulas and in different content markup languages or language proposals, because it was during the discussion on the representation of integration in OpenMath that we first introduced the notion of a compositional representation of generalized quantifiers into the discussion.

The example of a definite integral that we will use for this purpose is commonly written as

$$\int_0^x \sin x \; dx$$

Before considering how different formal languages express this mathematical formula syntactically, we will consider what a compositional analysis of the mathematical formula might look like. Table 7.1 summarizes an analysis that roughly corresponds to an OpenMath 1.0 representation of this formula, while the summary in Table 7.2 is perhaps closer to the

	Role	Component	Meaning
	parts	$[.,.]$	a binary operator denoting a closed interval
		0	a numeric constant, the integer 0
		x	a variable named x
(1)	syntax rule *application*	$[0, x]$	"application of operator interval to arguments 0 and x", or, the closed interval from zero to x
	parts	sin	a unary operator denoting the sine function
		x	a variable named x
(2)	syntax rule *application*	$\sin x$	"application of operator sine to variable x", or, the sine of x
	parts	$\sin x$	meaning of compound expression (2)
		x	a variable named x
(3)	syntax rule *abstraction*	$\lambda x.\sin x$	the unary function ($x \mapsto \sin x$) ("bind one variable, named x, and produce a unary function in that variable which returns the value of the body of the abstraction, namely the sine of that variable", or, the sine function in the variable named x)
	parts	$\int$	an operator denoting definite integration
		$\lambda x.\sin x$	meaning of compound expression (3)
		$[0, x]$	meaning of compound expression (1)
(4)	syntax rule *application*	$\int_{[0,x]} (\lambda x.\sin x)$	"application of the definite integral operator to the domain of integration, the interval from 0 to x, and to the integrand, the unary sine function," or, the integral of sine from 0 to x

Table 7.1. Compositional analysis of $\int_0^x \sin x \; dx$ as $\int_{[0,x]}(\lambda x.\sin x)$

MathML-Content representation and to an analysis given in [147, 148]. Table 7.3 summarizes a more linguistically motivated analysis of the same formula.

The example formula $\int_0^x \sin x \; dx$ is clearly a *compound expression* in the sense of the definition of the compositionality principle. We must therefore first ask what the *parts* of the compound expression are. In a simple analysis, the five main parts of this formula are

1. the integral operator,
2. the integrand, itself composed of
   (a) the function symbol sin, and
   (b) the argument of the function, the variable x

	Role	Component	Meaning
	parts	$[.,.]$	a binary operator denoting a closed interval
		0	a numeric constant, the integer 0
		x	a variable named x
(1)	syntax rule *application*	$[0, x]$	"application of operator interval to arguments 0 and x", or, the closed interval from zero to x
	parts	$\sin$	a unary operator denoting the sine function
		x	a variable named x
(2)	syntax rule *application*	$\sin x$	"application of operator sine to variable x", or, the sine of x
	parts	$\int$	a generalized quantifier denoting definite integration
		$[0, x]$	meaning of compound expression (1)
		$\sin x$	meaning of compound expression (2)
		x	a variable named x
(3)	syntax rule *generalized quantification*	$\int_{[0,x]} \sin x \; dx$	"generalized quantification with respect to the variable x, ranging over the domain $[0, x]$, using the generalized quantifier $\int$ – definite integration – and, as the body of the quantification, the sine of the quantification variable", or, the integral of the sine function from 0 to x

Table 7.2. Compositional analysis of $\int_0^x \sin x \; dx$ as $\int_{[0,x]} \sin x \; dx$

3. the integration variable,

4. the lower bound of the domain of integration, and

5. the upper bound of the domain of integration.

Of these, the upper and lower bounds may obviously be grouped together to form a single compound part of the expression, the interval of integration. This semantic grouping goes well with generalized notions of integration over arbitrary domains as in the notation $\int_D f$ for integrating function f over domain D.

A second grouping is less obvious, but will be crucial to understanding the analysis in Table 7.1, and its representation in OpenMath 1.0.

The integration variable can be understood to denote that the integrand is to be considered as a unary function in that variable, a concept that is variously expressed in mathematical notation as $(x \mapsto \sin x)$ or $(\lambda x. \sin x)$.

	Role	Component	Meaning
	parts	$\sin$ x	a unary operator denoting the sine function a variable named x
(1)	syntax rule *application*	$\sin x$	"application of operator sine to variable x", or, the sine of x
	parts	$\int \ldots d.$ 0 x	a generalized quantifier denoting integration a numeric constant, the integer 0 a variable named x
(2)	syntax rule *modification*	$\int_0^x \ldots d.$	"restriction of the general integration operator to the integration interval with lower bound 0 and upper bound x," or, a generalized quantifier denoting integration over $[0, x]$
	parts	$\int_0^x \ldots d.$ $\sin x$ x	meaning of compound expression (2) meaning of compound expression (1) a variable named x
(3)	syntax rule *generalized quantification*	$\int_0^x \sin x \; dx$	"generalized quantification with respect to the variable x, ranging over the domain $[0, x]$, using the generalized quantifier $\int$ – definite integration – and, as the body of the quantification, the sine of the quantification variable", or, the integral of the sine function from 0 to x

Table 7.3. Compositional analysis of $\int_0^x \sin x \; dx$ as $\int_0^x \sin x \; dx$

The latter rendering of this grouping unveils its most crucial aspect, namely, that the integration variable x is *bound* inside the integral, and that its scope is the integrand. Notations like $\int_D f$ denoting integration of a function f over some set D that one occasionally encounters in the scientific literature capture a similar notion, since for example sin and $(\lambda x. \sin x)$ are η-equivalent in the λ-calculus.

The fact that the variable x is bound within the scope of the integrand is underscored by the admittedly unusual choice of the upper bound for our example: the variable x occurs free in the upper bound of the integral, bound in the integrand, and as denoting the binding in the integration variable.[2]

[2]Not every mathematician agrees that an integral may have the name of the integration variable occur in the limits of integration. Indeed, the computer algebra system Maple signals an error when given a literal representation of this formula.

Putting the pieces together, we conclude that a compositional interpretation of the example expression is facilitated considerably by re-interpreting it as[3]

$$\int_0^x \sin x \; dx = \int_{[0,x]} (\lambda x. \sin x)$$

In this interpretation, the integration operator is thus interpreted as just one member of a large class of reduction operators that act on functional arguments over given sets or domains, a concept that is quite familiar from the mathematical field of functional analysis. The summation ($\sum$) and product ($\prod$) operators are two more operators of this very common kind of generalized quantifiers.[4]

Note that in this analysis of the compositional meaning of this example, summarized in Table 7.1, the handling of the binding and scoping of variables, a crucial component of first and higher order expressions like this one, has been delegated away from the specific operator (integration) to a single general-purpose ingredient of the language (lambda abstraction) whose specific task it is to express this notion, and this notion only.

The identification of *lambda abstraction* as the candidate concept for special semantic handling is a natural one for someone familiar with the lambda calculus, one of the fundamental theoretical tools of programming language research in computer science. It is also one that would come most natural to linguists working in the field of formal semantics.

However, as we can see in Tables 7.2 and 7.3, it is not the only one, and maybe not even the most intuitive one for a practicing mathematician or other scientist. Both OpenMath and MathML-Content have identified what we would term *generalized quantification* as the right concept for special semantic handling (and therefore special syntactic representation), with lambda abstraction just one of many special cases of this generalization. In this analysis, there are many operators that bind variables, and the concept for variable binding requires specification of such an operator to govern the bound variables.

Both lambda abstraction and generalized quantification are valid candidate concepts to add to the basic ingredient of application in a compositional analysis of first- or higher-order mathematical formulas such as our example. The compositionality principle as such does

[3]Note that Maple *does* accept and handle properly a literal representation of the right-hand side of this equation, but not one of the left-hand side.

[4]Indeed, any associative binary operator induces a member of this class just the way that addition and multiplication induce the summation and product operators. Hence, this class is quite large and open-ended.

CA system	$\int_0^x \sin x \; dx$
Maple	`int(sin(x), x=0..x)`
Mathematica	`Integrate[Sin[x], x, 0, x]`
REDUCE	`defint(sin x, x, 0, x)`

Table 7.4. $\int_0^x \sin x \; dx$ expressed in different Computer Algebra systems

not favor either one above the other. However, we will see in the next chapter (in section 8.6.2) that there are other criteria that indicate that the interpretation in Table 7.1 is indeed superior to those in Table 7.2 and Table 7.3.

Table 7.3, finally, is a sample analysis that uses the new concept of *modification*, a possible motivation for one of OpenMath's syntactic construct of OpenMath attribution objects. Here we see that the argument that we used to derive the necessity of providing special syntax for something like variable binding does not work in the opposite direction: the concept of modification can be represented by application just as well, but the compositionality principle does not mandate *orthogonality* of those concepts that are represented by special syntactic constructs in a language, and therefore does not prohibit the use of both application and modification simultaneously in the example analysis in Table 7.3.

7.2.1 Representation of Integration in Some Computer Algebra Systems

Table 7.4 lists the canonical representations of our definite integration example in a small selection of some current computer algebra systems.[5]

A common semantic task in a general purpose CA system is the evaluation of a complex expression involving free variables at specific points within the range of those variables, usually performed by substitution followed by simplification. Now consider asking any one of the systems above to substitute the constant π for the free variable x in the above examples.

Clearly, each system would have to treat differently all three occurrences of `x` in these expressions: in the integrand, `x` may not be replaced by π because it is protected by another occurrence of `x` which declares it bound in the integrand; that occurrence of `x`, of course, may not be replaced, either, but needs to be interpreted as the definition of scope for that

[5]Most Computer Algebra systems can represent the example in both the canonical way shown here and in an alternative syntax that translates the analysis shown in Table 7.1: $\int_{[0,x]}(\lambda x.\sin x)$.

variable; only the occurrence of `x` that denotes the upper bound of the integration interval may be replaced by the constant π.

Both REDUCE and Mathematica use the same syntax in this example (which involves variable binding) as they use for ordinary application of a function to four arguments. As we saw earlier, this means that their representations are non-compositional.

Only the Maple syntax provides a clear way (in principle) to distinguish between range, integrand, and integration variable that does not necessarily require knowledge about the operator. As the Ctadel language [144] shows, if used for this purpose only, this kind of syntax can be compositional despite a nesting of parentheses in the expression that is totally unrelated to scope. In practice, however, there is evidence that Maple does not use this syntax in the rigorous manner that would be required to qualify as compositional. For one, Maple can interpret equations of the sort used as a third argument in the example as a simple application of the equality predicate to its two arguments, which allows their syntactic appearance anywhere in an expression where a logical formula is allowed, even if no variable binding is intended. Secondly, the specific example given in Table 7.4 actually causes Maple to signal an error that essentially says that the integration variable must not appear on the right-hand side of the equation `x=0..x`, which means that Maple does not actually interpret this construct as binding a variable. This construct in Maple is therefore also non-compositional.

As we can see, compositionality appears to have played little role in the design of user languages for general-purpose Computer Algebra systems [2].

Nevertheless, these systems will likely give the right answers to the substitution task,[6] but correctness in all cases comes at a significant price. Each of these systems needs to tag their respective names for the integration operator as what LISP would call an "FEXPR" – an operator that handles the interpretation of its arguments on its own rather than leaving their interpretation to the system. In REDUCE, for example, the integration operator is tagged with its very own substitution and simplification routines, barring the generic routines from touching its arguments before handing them over to the routines that are specific to integration.

[6] With the exception of Maple, which balks at an appearance of the integration variable in an integration bound.

The key here is that *these operators handle the interpretation of their arguments on their own rather than leaving their interpretation to the system* – which is just another way of stating that they are not being handled in a compositional manner by these systems. From a software engineering point of view, such a non-compositional design does not scale well. For each new operator (summation, product, differentiation, roots extraction, limits, and so on *ad infinitum*), its very own syntactic rule would need to be declared were compositionality to be claimed.

Not that this problem is entirely new. In LISP systems, a similar argument involving compilation (another form of semantic analysis) resulted in the need to start adding "syntactic sugar" to the language in the form of MACROs rather than FEXPRs, as a compromise between legibility and easier writing of software on the one hand and compositionality, extensibility, and thus compilability on the other.

7.2.2 Representation of Integration in OpenMath

OpenMath has gone through several revisions over the years. The first draft for OpenMath [19] still represented our example in a way that was very similar to the REDUCE and Mathematica representations quoted above:

```
(&int. (sin x) x 0 x)
```

Apparently, compositionality of the representation was not yet a major concern at that point of the OpenMath discussion; OpenMath is thus firmly rooted within the tradition of computer algebra systems. It is all the more remarkable then that OpenMath eventually broke with this tradition, albeit after a long, intense, and sometimes even bitter struggle, and it is all the more interesting to understand why it did.

The modern representation of definite integration in OpenMath

Compare the OpenMath starting point with the modern OpenMath [10] representation for our example, published about six years after the initial attempt:

$$\begin{aligned}&\textbf{application}(\texttt{defint}_{\texttt{calculus1}},\\ &\qquad\textbf{application}(\texttt{interval}_{\texttt{interval}},\ 1,\ x),\\ &\qquad\textbf{binding}(\texttt{lambda}_{\texttt{fns1}},\ x,\ \textbf{application}(\texttt{sin}_{\texttt{transc1}},\ x)))\end{aligned}$$

This representation is a mix of our previous analyses of the compositional structure of our example formula in Table 7.1 and Table 7.2. OpenMath clearly distinguishes syntactically between function application (**application**) and generalized quantification (**binding**), and it uses the concept of integrating a unary function over an interval to represent this kind of definite integration.

The representation for lambda abstraction itself in this version of OpenMath follows the spirit of the alternative compositional analysis of integration as a generalized quantifier in Table 7.2.

Representations of definite integration in older OpenMath versions

Let us now take a look at how the discussion on representing definite integration developed in the OpenMath group from the traditional beginning to the almost revolutionary current version.

In December 1996, in the course of a lively and controversial discussion at the Dublin OpenMath workshop, we first argued that the traditional representations of this example that were being proposed for OpenMath at that point violated the Compositionality Principle. In particular, we observed that it was a bad idea to have an expression where one argument was within the scope of a second argument (a variable) and a third argument was outside the scope of that second argument. We thus proposed using a lambda expression to factor out the concept of variable binding from the representation of integration. The arguments brought forward by the supporters of our proposal eventually compelled a vote in favor of a representation of our example in the conceptual form

```
int( lambda(x, sin(x)), 0, x)
```

instead of one of the classic representations (and indeed a vote to include lambda as one of the core OpenMath operators).

The observation that operators of the integral and series kind abound in mathematics, making necessary and useful a general mechanism for marking variable scopes, together with the realization that scalable user extensibility of the OpenMath vocabulary was a *conditio sine qua non*, became the decisive argument for this solution: it was felt that any variation on the classic representations would have required special treatment for each new such operator,

and it was clear that that would severely limit the ease of extending OpenMath with new operators of this very important class.

In the summer of 1997, G. Gonnet and A. Diaz presented an SGML-based draft of an OpenMath specification that took this decision into account.[7] With respect to the goal of compositionality, and compared with our analysis in the previous section, their solution still had a problem in that `lambda` was treated as a normal mathematical operator and no special syntax for variable binding was provided. Consequently, in [2], we still came to the conclusion that that version of OpenMath did not yet fully support a compositional representation of our example, though we remarked that an easy fix was available by moving the symbol `lambda` from the `core` dictionary of common mathematical symbols to the `meta` dictionary of special symbols, thus causing the `lambda` symbol (and hence lambda abstraction) to be treated as special syntax.

The Esprit Project OpenMath instead decided to take a more radical final step towards compositionality. Their redesign of OpenMath, which introduced the new syntactic binding constructor in addition to the classic function application one, was explicitly based on the adherence to the Compositionality Principle as a decisive argument for the proposed change.

7.3 Non-Compositional Constructs in the Knowledge Interchange Format 3.0

In [2] we first reported our finding that version 3.0 of the Knowledge Interchange Format (KIF) has two generalized quantifiers whose semantics are defined in a non-compositional manner: `the` and `setofall`.

In a weak version, this judgment is not necessarily problematic as such. According to [23], for example, the standard mathematical logic interpretations of the existential and universal quantifiers via variable assignment substitutions are non-compositional. However, [23] shows how this can be fixed, and that method applies directly to the interpretation specified for these quantifiers in the KIF specification in either version. The KIF representations for existential and universal quantification are thus "weakly" non-compositional.

[7] This draft is no longer available on the web, unfortunately.

Our claim with respect to the two KIF operators mentioned in [2] is much stronger. In this case we claim that their definition is flawed beyond repair by the means outlined in [23], or in other words, that syntactic changes are needed in order to be able to apply the method outlined in [23] for making them compositional.

7.3.1 KIF's `the` and Scalability

The definition of the interpretation of KIF 3.0's syntactic construct `(the` τ ϕ`)` reads as follows:

> "The referent of a designator with term τ as first argument and sentence ϕ [as second argument] can be one of two things. Consider all versions v' of [variable assignment] v with respect to the free variables in τ. If there is at least one version v' that makes ϕ true and the semantic value of τ is the same in every v' that makes ϕ true, then the semantic value of the designator as a whole is that value. If there is more than one such value, the semantic value is $\perp$." [93]

What makes this definition non-compositional is that τ is an arbitrarily deeply nested term whose free variables need to be extracted as one of the parts referred to in the interpretation: the list of free variables is thus an implicit rather than explicit part of the compound `the` expression, which makes it non-compositional.

Furthermore, the interpretation of `the` is non-compositional because it manipulates directly the "hidden" feature of interpretation, variable assignments.

In [2] we argue that non-compositional semantics do not scale. In the case of `the`, it is readily apparent that this is true: adding other operators like `the` would obviously require redefining any interpreter for this language, because the interpretation would manipulate internal features (such as variable assignments) of the interpreter.

7.3.2 KIF's `setofall` and Correctness

In [2], we also argue that compositional semantics are required for correctness. This is particularly well illustrated by the non-compositional interpretation for KIF 3.0's `setofall` operator.

The `setofall` operator in KIF 3.0 is meant to be used for formulas such as $\{x \mid x^2 = a\}$, and is its direct translation:

```
(setofall ?x (= (expt ?x 2) a))
```

Let us quote the definition of the interpretation of `(setofall` τ ϕ`)`:

> "A set-forming term with the term τ as first argument and the sentence ϕ as second argument denotes the set of objects in the universe of discourse with the following properties.
> (1) The object must be the semantic value of τ for some version v' of v that makes ϕ true.
> (2) The object must be *bounded*." [93]

Our criticism of the "`the`" operator apply to this definition as well, but there is more. To see why, let us consider this alternative formulation of Fermat's Last Theorem as an example:

> For an integers n greater than 2, the intersection of the set of all $x^n + y^n$ such that both x and y are positive integers, and of the set of all z^n such that z is a positive integer, is empty.

This formula expresses the same more concisely:

$$n \in \mathbb{N} \wedge n > 2 \Rightarrow \{x^n + y^n \mid x, y \in \mathbb{N}\} \cap \{z^n \mid z \in \mathbb{N}\} = \emptyset$$

The obvious, canonical translation to KIF 3.0 is this:

```
(=> (and (integer ?n) (> ?n 2))
    (empty
      (intersection
        (setofall (+ (expt ?x ?n) (expt ?y ?n))
          (and (integer ?x) (integer ?y)
            (> ?x 0) (> ?y 0) ))
        (setofall (expt ?z ?n)
          (and (integer ?z) (> ?z 0))))))
```

Syntactically, this exactly mirrors our example formulation of Fermat's Last Theorem, but semantically, the KIF 3.0 interpretation gives the wrong result. The KIF 3.0 statement above, interpreted in accordance with the definition of `setofall`, has truth-value `false` where we know from the work of mathematicians over the centuries, culminating in Andrew Wiles' recent proof, that it should have a truth-value of `true`.

We can see why the KIF interpretation of the above statement produces *false* rather than *true* by considering the second `setofall` sub-expression in the KIF 3.0 formulation of our example, `(setofall (expt ?z ?n) (and (integer ?z) (> ?z 0)))`. The definition for `setofall` states that it is to be interpreted as the set of all objects in the universe of discourse that are a semantic value of `(expt ?z ?n)` *for some version* v' *of* v that makes `(and (integer ?z) (> ?z 0))` true. But this includes arbitrary variable assignments for `?n`, including `n` = 1, and therefore that particular `setofall` expression describes the set of all integers greater than 0. Obviously, therefore, the intersection of the two sets described by the `setofall` expressions in our example is non-empty!

Note that it is possible to re-phrase the KIF expression to match the intended meaning of the sentence:

```
(=> (and (integer ?n) (> ?n 2))
    (empty
        (intersection
            (setofall ?w
                (exists (?x ?y)
                    (and (integer ?x) (integer ?y)
                         (> ?x 0) (> ?y 0)
                         (= ?w (+ (expt ?x ?n) (expt ?y ?n))))))
            (setofall ?w
                (exists (?z)
                    (and (integer ?z) (> ?z 0)
                         (= ?w (expt ?z ?n) )))))))
```

Note how this representation relies on the fact that the variable `?n` is not bound by `setofall` if it only appears in the second argument but not the first, and how this makes it necessary to explicitly bind the variables `?x`, `?y`, and `z` using an existential quantifier. Note also that this is syntactically very different from the mathematical formula that the `setofall`

construct was originally meant to represent, which argues again for a design error in the case of `setofall`.

7.3.3 Compositionality and Correctness

We thus have an example where a violation of the compositionality principle is the direct cause for a serious error in a content communication language. We can also see that this is not a fluke of a single example in a single language but a serious problem for content markup language design in general is underscored by the fact that exactly the same kind of errors can be produced by KIF's `the` construct, and that exactly the same conceptual representation error was made about six years later and quite independently by another large language specification group in the specification of the `max` and `min` operators of MathML 1.0.

In effect, what both the `the` and the `setofall` constructs' interpretations are saying (and what MathML's `min` and `max` featured as an optional representation) is this:

> *First,* extract all the free variables from the first argument, a term, and bind these variables in the scope of the compound expression headed by the operator.
> *Second,* let the bound variables range over the entire universe of discourse and determine for which assignments to the bound variables the second argument, a sentence, is true.
> *Finally,* determine the semantic value of the entire expression from the set of objects (i.e., variable bindings) that satisfy the sentence.

What makes these definitions non-compositional is that neither the list of variables nor the fact that they are bound within the scope of the compound expression is made explicit in the syntactic construct representing the concept.

Thus, what makes these interpretations dangerous is that they do not make a clean distinction between the bound and free variables in one of the argument expressions. In our example, we needed the ability to declare that only `z` was bound by the `setofall` operator (and therefore, only versions of v wrt. variable `z` were to be considered: "`z` ranges over the integers"), while variable `n` was to be treated as a free parameter of the set.

MathML 1.0 similarly explicitly allowed dropping the explicit list of bound variables and to extract it from a constructor expression instead. This "feature" was deprecated by version

2.0 after we pointed out the problem using a similar example as the one used above with `setofall`.

7.3.4 Improving the Knowledge Interchange Format

A better representation of the `setofall` concept would therefore have been a three-argument operator `setofall`, the first argument of which would have been a list of variables bound by the operator (thus treating it syntactically just like any other generalized quantifier), and the second and third arguments of which would have been the same as the first and second one of the KIF 3.0 operator of the name. The definition of the interpretation of the new operator would have mentioned explicitly the fact that variable binding was in fact taking place, and listed the variables bound in this scope. Other than that, it would have remained the same except for specifying exactly with respect to which variables v' was allowed to be a version of v.

Such a representation of `setofall` would have been compositional in that it lists all the ingredients necessary for defining the interpretation as explicit top-level parts of the compound expression. It still falls short of the goal, however, in that its interpretation expresses again, explicitly, the concept of variable binding that is already prototypically expressed in KIF 3.0 in the definitions of the `lambda` and `kappa` abstraction operators.

Thus, KIF 3.0 provides us with a valuable object lesson in the importance of compositionality both for the scalability and for the correctness of content communication languages.

7.4 Compositionality in the Knowledge Interchange Format Draft Standard

Given the above-mentioned bugs in KIF 3.0, it is hardly surprising that the operators `setofall` and `the` disappeared in the draft proposed ANSI standard for KIF [94] about six years later. Indeed, except for the two quantifiers `forall` and `exists`, all generalized quantifiers and variable-binding operators have disappeared, including, unfortunately, `lambda` and `kappa` abstraction.

While this eliminates the correctness problems that were the consequences of non-compositional treatment of some generalized quantifiers, and while this may make dpANS-

KIF technically compositional, the price that is paid, a serious reduction in expressiveness of the language, is high.

7.4.1 Generalized Quantifiers and dpANS KIF

Defining common mathematical concepts such as integrals or series becomes difficult or impossible without such constructs.

As we have seen, these operators are generalized quantifiers: they operate on functions with one or more variables and in general have as values such functions, or in some cases, individual constants. For example, as the typical integration technique of variable substitution makes clear, the integral operator binds the integration variable, and similar observations may be made for other generalized quantifiers.

We saw above that one compositional manner of representing such generalized quantifiers is to treat them as operators on function space, perhaps adding ranges over which they operate as arguments in the case of definite integration. For example, $\int_a^b x^n dx$ would be treated as $\int_{[a,b]}(\lambda x.x^n)$. This treatment factors out the variable-binding issue that was the main source of errors in the `setofall` example above, to the underlying principle of λ abstraction.

Since the `lambda` abstraction operator is no longer available in dpANS-KIF, however, this solution is not supported directly in this language. Indeed, the only possible way of defining a function in the new version of KIF is by using `deffunction`. However, `deffunction` may only be used at the top-level, so this doesn't really help if we want to, say, express general integration or differentiation rules where we may need to transform one functional argument into another:

$$\int f'(x)g(x)dx = f(x)g(x) - \int f(x)g'(x)dx$$

The problem is that the only way to introduce a clean, variable-binding operator into a language is by integrating the concept of abstraction, and a syntactic feature that expresses it, into the interpretation of the language. Only then are we guaranteed that bound variables will turn into free variables whenever a sub-expression is extracted from within the scope of a generalized quantifier.

7.4.2 Backquote

Thus, it is impossible, for example, to use dpANS-KIF's quote and backquote[8] facilities to specify the semantics of variable-binding operators, which could otherwise be used to express the integration example above as, say,

```
(integral ^(expt ?x ,?n) '?x)
```

In this example, the integration variable is protected from being assigned a value outside the scope of the quote operator, and thus effectively bound, but nothing stops the unwary from crossing the integration variable's scope boundary with a reference to the same variable thusly:

```
(if (= ?n '(floor ?x)) (integral ^(expt ?x ,?n) '?x))
```

In this example, if unification of the `if`-clause causes `?n` to be replaced by (unified with) `(floor ?x)` in the integral expression, say, the exponent in the integrand, which was independent of the integration variable, becomes a function of `?x` with potentially completely wrong evaluations of the integral as a result.

Note that the only reason why this kind of an error is unavoidable in an environment of quoted expressions is that the interpretation does not look inside them – must not, in fact, look inside them – and thus can never guarantee that variable scopes expressed inside a quoted expression will be honored. We therefore argue that it is effectively impossible to correctly define the semantics of variable binding by relying on KIF's quote and backquote mechanisms, there being no effective way of avoiding such scope-crossing interpretation errors as in the example.

In other words, backquote and quote are a non-compositional feature with great potential for error in variable scoping. The common technique of *reification* via quote and backquote mechanism that is commonly used in AI knowledge representation formalisms in order to circumvent limitations of first-order logic can thus be seen to be useless when it comes to representing higher-order mathematical formulas.

If it is therefore impossible to correctly express the meanings of commonplace mathematical operators and generalized quantifiers such as integration or summation, dpANS-KIF

[8] KIF 3.0's (and Common LISP's) backquote syntax has been replaced by a caret syntax in dpANS KIF.

hardly qualifies as a general-purpose language for communicating knowledge among agents, since agents that do understand symbolic integration are well-known and available off the shelf: general-purpose computer algebra systems being just one class of such systems.

Thus, dpANS-KIF shows how a well-meaning concentration on first-order logic as a means of begging the compositionality question entails a major reduction in the expressive power of such a language that makes it effectively impossible to implement compositional representations for generalized quantifiers within the language. And as we have repeatedly seen in our examples, non-compositionality implies the possibility of incorrectness; in dpANS-KIF, this has been demonstrated for the backquote syntax.

7.5 The Mathematical Markup Language and Compositionality

As mentioned earlier, we have applied the compositionality principle successfully[9] to language improvements for the content markup part of the Mathematical Markup Language (MathML).

We were the first to point out the importance of a compositional content markup language component of MathML, especially in the context of variable bindings, in a posting to the public MathML mailing list in late 1997.

While the direct impact that our introduction of this concept into the discussion might have had on the concrete development of MathML-Content is hard to judge because almost all the design discussion went on in closed mailing lists and meetings, it is definitely fair to say that the MathML-Content language design improved continually over the years with respect to the compositionality criterion.

However, there were two cases where we successfully made specific recommendations that helped MathML on this track. One of these we already mentioned in the context of our discussion of the `setofall` construct in KIF 3.0, where we mentioned that a non-compositional "feature" of MathML-Content is now deprecated because we showed, again in a posting to the public MathML mailing list, that it could easily lead to incorrect interpretations.

[9]The editors of the Mathematical Markup Language (MathML) recommendation documents have acknowledged our contributions to their effort for several years now.

A more substantial concrete and traceable contribution that we made to the MathML-Content language design was a special case of the second design recommendation that we extracted from the compositionality principle in section 7.1.4 above, namely the recommendation that syntactic constructs that represent semantic features which occur in an open-ended number of contexts be defined as general syntax rules rather than syntax rules available only in special contexts. In this particular case, MathML-Content originally restricted the use of *qualifiers*, among them most prominently those used for the specification of bound variables, to a finite number of special cases listed in the recommendation. In a posting to the public MathML discussion forum, we successfully argued that this restriction should be lifted and that qualifiers, especially those used for variable binding, should be defined as general-purpose syntactic features of MathML-Content instead.

7.6 Outlook: A Taste of Things to Come

For the rest of this chapter, we will get a taste of things to come where further research into applications of the linguistics parallel in general, and of the compositionality principle in particular, can have an important impact on the design of further aspects of knowledge communication and content markup.

7.6.1 Compositional Cross-References

When compared with our "Objectives of OpenMath" [1], there is one particularly obvious omission in the current OpenMath, an omission that has been under discussion for many years without a final resolution to this day. We are talking about the lack of support for cross-referencing (or sharing) of sub-expressions.

Looking for correspondences in linguistics, we quickly realize that human languages support a clean distinction between at least three kinds of cross-references. The first of these, exemplified by reflexives in some languages and traces in all human languages, is local to a sentence (the unit of speech that we will deem equivalent to an OpenMath object here) and obeys strict scoping rules within the sentence structure. The second, typically "implemented" e.g. as pronomials, has cross-sentential reference capabilities and is subject to dynamic re-interpretation, but is local to a conversation. A third, "implemented" by

naming in all languages of the world, establishes references that are potentially available across even long-term interruptions of communications.

Clearly then, if we accept the premise of parallels between the structure of content markup languages and linguistics, OpenMath is lacking the second of these three kinds of referential structures. Interestingly, it has long been argued that this particular type of linguistic structures violates the Compositionality Principle and requires new "dynamic logics" [25], and indeed it is partly due to the difficulty of establishing simple rules for interaction between such variables and variable binding that OpenMath has been reluctant to adopt a standard here. Recently, however, [149] claim success in defining a compositional dynamic lambda calculus which they use to derive formal semantics for linguistic phenomena of this type. Thus, their work might well serve as a valuable starting point for remedying this particular short-coming of OpenMath, especially seeing that one of the authors of that paper is very active in the OpenMath effort.

7.6.2 Pragmatics

Some efforts are currently investigating ways to extend OpenMath with mechanisms for delivering "commands" (i.e. specifications of which actions are to be performed on the OpenMath objects communicated to a process). Indeed, early drafts of OpenMath included a concept of "verbs" versus "nouns" (based on a similar distinction found in Maple).

In the context of linguistics, this aspect of communication is typically studied under the heading of "pragmatics," a branch of linguistics that is very likely to provide some crucial insights into a good design for such an OpenMath extension.

Let us, for the sake of brevity, just concentrate on the analogy suggested by older OpenMath distinctions between "verbs" and "nouns," and, as a first approximation, let us take as a homologue of a noun phrase an OpenMath object that has as a "head" one of the currently available kinds of OpenMath objects ("nouns"), and let us consider as a parallel of the verb phrase OpenMath structures with a "head" element from a "pragmatic" extension of OpenMath.

There are several lessons that we can immediately draw from this. One rather obvious one is used extensively in Maple: almost all nouns (and noun phrases) can canonically be turned into verbs (and verb phrases), and vice versa. Another less obvious lesson, indeed

a possibly quite controversial one, is that just as verb phrases may (and do) contain noun phrases as arguments, so can (and do) noun phrases have verb phrases as arguments, to arbitrary depths of mutual inclusion. In other words, if an OpenMath extension defines commands, these commands should (eventually) be allowed not just at the top level, as most current proposals suggest, but also at arbitrary nesting depth. In particular, the homologue of "direct speech" would require that such complex commands be communicable objects in their own right.

In this case, the compositionality principle would lead to a suggestion that the pragmatic and semantic layers of such a language be distinguished syntactically.

7.6.3 Advanced Treatment of Integration

Finally, let us return to a small variation of the example that we used at the beginning of this chapter, in order to see how well Vannevar Bush, whom we quoted earlier as saying that mathematical notations "grew like Topsy and [have] little consistency" [45], understood the facts of mathematical notation, and in order to discover a few more linguistic phenomena that turn up in its analysis.

Let us consider the equation

$$\int dx = x$$

Earlier, we argued that the integration variable x is bound by the integration operator. Taken literally, this would mean that $(\int dx = y)$ would be just as correct, but our intuition tells us that that is not so. The variable x, even though bound in the integral, is somehow also "exported" from the integral in apparent violation of the notion of variable binding.

A linguistically inspired solution to this conundrum that we would like to propose is that the name x is more than just the name of a variable: the choice of the name x says "we're moving in a space of functions in (the variable) x". Thus, the function space indicated by the name of the variable is the implicit *topic* of this equation, and is actually "imported" into the integral from the global environment.

Unlike "exports", such "imports" into a substructure are a common phenomenon in compositional linguistic analyses as they can be modeled as argument passing. Again the compositionality principle allows us to find a reasonable approach to an otherwise rather vexing problem.

Thus, the two occurrences of identical names of variables denote the common notion of "functions over a common space" while at the same time denoting two different variables with different scopes in a semantic interpretation.

In other words, a variable in mathematical notation often plays a dual role, leading to easy confusion when trying to devise an unambiguous formal representation. Disambiguation of such multiple roles is something that people excel at in everyday language use, so it should be no surprise that such ambiguities abound in mathematical notation: mathematical notation is made by intelligent people for communication with other intelligent people, and mathematicians hate writing down (or, for that matter, reading) what can be "easily" (that is, with the intelligence of an informed mathematician) inferred.

To illustrate the amount of implicit information left out of mathematical notation, let us take a closer look at this example again:

$$\int dx = x$$

may, similarly to our previous analysis of definite integration, be rewritten in a less concise manner as

$$\int_{x_0}^{x} dx = x + C(x_0)$$

and then to an even more verbose

$$((x, x_0) \mapsto \int_{x_0}^{x} x^0 \; dx) = ((x, x_0) \mapsto x + C(x_0))$$

By now it should have become obvious that the linguistic analysis of mathematical formulas is by no means a trivial task; however, it should also be clear that it is quite a useful and rewarding one.

Given the huge amount of information that is obviously left implicit (i.e. unmentioned) in the notation, and given the complex interplay of cross-links between elements of the notation, the linguistic notion of "traces" and "co-reference" will certainly play a crucial role in such analyses, and their conceptual counterparts are probably very real candidates for inclusion in future versions of OpenMath. In addition to these two elements of linguistics, this simple example also brought us into contact with the extremely important linguistic concept of topicalization.

7.7 Conclusions

Finding a good design for a general-purpose knowledge interchange or content communication language is surprisingly hard, as witnessed by the mistakes made even by professional computational logicians like the designers of KIF, and by the fact that several years later, almost exactly the same kinds of design errors were made in the definition of MathML 1.0 and later deprecated when we pointed them out in MathML 2.0. One of the hardest problem in this context appears to be the integration of generalized quantifiers and first- or higher-order mathematical operators into such a language.

Research into the semantics of human languages shows that scoping issues are handled by their very own general-purpose syntactic constructs and associated general-purpose interpretation mechanisms for bindings, presumably due to the underlying compositionality principle that, as we have argued, requires such mechanisms. The same argument, as we have seen, applies to the design of content communication languages.

We have shown that many languages have failed to do so, and shown through some examples that this failure can have disastrous effects on the expressiveness, scalability, and most importantly, correctness of representations of the meanings of common mathematical operators in these languages.

Chapter 8 discusses an important linguistic design and analysis tool that crucially depends on the compositionality property, namely, the concept of a categorial grammar and type theory. There we will see that the compositionality principle suggests treating yet more constructs as semantically, and hence, syntactically special.

CHAPTER 8

CATEGORIAL SEMANTICS

8.1 Introduction

In Chapter 7 (see also [2]) we argued that content markup languages need a compositional design. Mathematically, the compositionality principle has been modeled in the literature as a requirement for the existence of an interpretation that acts as a homomorphism from a syntactic algebra to a semantic algebra. From a more practical computer scientist's perspective we have noted that it implies that

> "a language for expressing semantics for exchange among computer programs [...] define[s] a basic interpretation procedure – how to interpret the parts, and how to interpret the combinations of parts. Compositionality is important because it means that this interpretation algorithm can be defined completely with respect to how to interpret combinations of parts once you have interpreted the parts. Defining a skeleton interpretation involves defining the structure of the lexicon and that of its entries, but no specific lexicon entries should need to be referred to in its definition. Thus, compositionality means that one is able to provide a firmly grounded scaffolding from which one can work to extend the scope of [the] language by adding only lexical entries, while keeping the scaffolding intact." [2]

Consequently, the challenge for the practical designer of OpenMath and similar languages is to construct such a "skeleton interpretation" for their language. This exercise can be a valuable design tool for content communication languages, as we will show in this chapter by using it on OpenMath (which we have seen in Chapter 7 is compositional and thus amenable to such a construction).

A *skeleton semantics* for a compositional language, as we termed it, i.e. a compositional semantics that respects the syntactic structure of a language, has been studied in linguistics

under the general heading of "categorial grammar," and it is in this sense that we use the term "categorial" in the following. [1] The interested reader is invited to consult M. Moortgat's survey of *Categorial Type Logics* [24] for a justification and history of this terminology, and for an in-depth treatment of both the notions and the notations we use in the following.

8.2 Problems of Formal Semantics for Content Markup

The specification of a semantics for content markup languages like MathML-Content or OpenMath appears to be a thorny issue. Both languages have essentially side-stepped the issue by declaring their semantics "informal" and thus, open to interpretation.

In the case of OpenMath, however, two proposals for OpenMath type systems were published simultaneously with Version 1.0 of the Standard [10], in order to provide a more ("Strong" OpenMath [32]) or less (Small Type System [31]) detailed semantics for OpenMath objects. Thus, the OpenMath stance appears to be that many type theories exist, all of them equally valid, and since any formal semantics induces a type theory, any number of such semantics for OpenMath must be allowed to co-exist peacefully.

I am not aware of any similar project for MathML Content markup; the MathML philosophy appears to be that its semantics is informal by definition and no formal semantics is desired (or maybe even possible).

On the other hand, developers of another knowledge communication language, the Knowledge Interchange Format (KIF) [93], invested considerable energy in providing a fully formal semantics (but not type theory) for their entire language. Ontologies based on KIF have included "engineering mathematics" [150], making KIF directly comparable with MathML and OpenMath despite its quite different designated application area of intelligent agent communication.

8.2.1 Completeness vs. Extensibility

The main reason for avoiding the specification of a formal semantics for a content markup language, for example in the form of a formal type system, appears to be grounded in two

[1] I am indebted to O. Caprotti, M. Kohlhase, and L.J. Kohout as well as two anonymous reviewers of a paper containing some of this material that we submitted to *International Congress on Mathematical Software 2002*, for insightful discussions of some of the material presented in this chapter.

conflicting demands on such a language and its semantics. On the one hand, a content markup language needs to be extensible in order to allow the marking-up of novel concepts in the language, but on the other hand a formal semantics for a content markup language needs to be complete in the sense of providing an interpretation for all possible legal sentences in that language.

8.2.2 Power vs. Efficiency

Another reason appears to lie in the fact that different type theories make different choices when it comes to weighing type inference power (and thus the set of constructs that are recognized as valid) against automatic validation speeds (or indeed decidability). Defining too weak a formal type theory, it can be argued, would run the very real danger of prohibiting the use of even quite common scientific formalisms, but would allow one to require its implementation by "conforming" applications. Use of too powerful a formal type theory, on the other hand, might allow one to formalize the semantics of a wide range of conceptual structures, but the formal proof theory for such a type system might be prohibitively expensive to implement – if a decision procedure should exist at all.

8.2.3 Categorial Semantics and Categorial Types: A Compromise

The concept of a categorial semantics in general, and of a categorial type system in particular, tries to provide a good compromise between the two extremes of both these dimensions.

A categorial semantics is *complete* with respect to the syntactic structures provided by a content markup language, but it is at the same time *extensible* with respect to the lexical ingredients of the language.

By restricting itself to the *structural* semantics of a content markup language and leaving the *lexical* semantics unspecified, a categorial semantics can be kept simple enough to be implemented easily (at least in principle – its actual implementation may not actually be required of a conforming application.) By providing a clean conceptual interface to "external" type theories on (a subset of) the atoms of the language, it acknowledges the fact that different formal semantics are possible and, indeed, useful, but by providing a single common

structural semantics for the language it makes sure that any two such semantics agree at least on the basic meanings of an expression.

8.3 Origins of Categorial Semantics

As mentioned above, we have "stolen" the concept of a categorial semantics from Formal Semantics, a branch of Formal Linguistics. In Formal Semantics, sometimes also known as Montague Semantics, categorial types are a fundamental tool for studying the structure of the semantics of language. More specifically, *Categorial Type Logics* (thus the title of a chapter [24] in a recent survey of the field, the *Handbook of Logic and Language* [25]) have been studied extensively in this field.

We will therefore adapt a formalism from this field, known as Lambek calculus, for the purpose of illustrating its usefulness in the much more recent field of content markup language design.

8.3.1 Compositionality and Lexicalism

A categorial semantics for a language can only exist if it obeys the twin principles of *Compositionality* and *Lexicalism*. A strong version of the compositionality principle can be expressed mathematically as the requirement that the semantics be a homomorphism between the syntactic and semantic algebras of the language. Lexicalism, in addition, requires all atoms of the language to be assigned a type or semantics exclusively via a *lexicon*, leaving the semantics of compound expressions of a compositional language up to the homomorphic images of the language's syntactic constructors.

Transferring these concepts to the field of content markup language design, the lexicon is a mechanism to provide a formal semantics for those parts of the content markup language that are extensible (in OpenMath, for example, the *OpenMath content dictionaries* and their OpenMath symbol entries), while the homomorphism between the syntactic and semantic algebras provides an interpretation of those parts of the language definition that are intended to remain fixed over time, namely, its syntactic constructors.

8.3.2 The Syntactic Lambek Calculus

The Lambek calculus[2] is originally one of syntactic categories (hence the term *categorial*), for purely syntactic purposes. However, due to the homomorphism between syntax and semantics postulated by the strong compositionality principle, it translates in a straightforward manner to the semantic level.

In the Lambek formalism, the lexical category of a word such as an intransitive verb such as *loves* might be specified as $np/s\backslash np$, denoting, in the syntactic algebra, "a sentence (of syntactic category s) lacking a noun phrase (of syntactic category np) on its syntactic left and a noun phrase on its syntactic right," and thus commonly interpreted in the semantic algebra as of type $NP/S\backslash NP$, with S and NP the types (or semantic categories) of sentences or noun phrases, respectively.[3]

The syntactic operation of concatenation is denoted by "$\bullet$" in the Lambek calculus. Thus, the sentence *Peter loves Mary* might be categorized in the syntactic Lambek calculus as $np \bullet (np\backslash s/np) \bullet np$ (the concatenation of a noun phrase, a transitive verb, and a noun phrase, in that order), and the semantic homomorphic image gives an interpretation of $NP \bullet (NP\backslash S/NP) \bullet NP$.

Finally, the syntactic Lambek calculus includes categorial inference rules such as [4]

$$\frac{B \bullet (B\backslash A)}{A}\,\backslash E$$

and

$$\frac{(A/B) \bullet B}{A}\,/E$$

[2]This material is loosely based on [24].

[3]In Montague Grammar, we usually find $S := t$ (with t denoting the set of truth values) and $NP := ((e \to t) \to t)$ (with e denoting the set of elements of the universe of discourse). This interprets a sentence as a statement that is true or false (or true or false to some degree, perhaps) and the referent of a noun phrase as the set of all properties that hold of it.

[4]The common proof tree notation $\frac{\alpha}{\beta}$ is used here to mean "β can be inferred from α". Inferences can be annotated with the names of the inference rules they are examples of, as in $\frac{\alpha}{\beta}\,Rule$ with inference rule *Rule*.

with categorial variables A and B [24], which allow one to prove, for example, that

$$\frac{\dfrac{np \bullet ((np\backslash s)/np) \bullet np}{np \bullet (np\backslash s)}\, /E}{s}\, \backslash E$$

with concrete syntactic categories np and s substituted appropriately in obvious ways for the categorial variables A and B in the above categorial inference rules. This provides a model for the syntactic parsing process for a language, where the ability to infer s corresponds to acceptance of a sequence of syntactic categories as constituting a valid syntax tree for a sentence of the language (the lexicon provides a mapping from the words of the language to their syntactic categories).

8.3.3 The Semantic Lambek Calculus

We are, however, more interested in the homomorphic image of the Lambek calculus in the semantic algebra, where we have S and NP as the semantic interpretation of the syntactic categories s and np, resp.

The syntactic concatenation operator $\bullet$ is interpreted semantically as a form of *application* (also denoted as $\bullet$). More precisely, if F is the semantic category of unary functions with domain A and range B, then $F \bullet A := B$. Thus, if $f \in F$ and $a \in A$, then $f(a) \in F \bullet A$: the semantic interpretation of the $\bullet$ notation is application at the type level.

Similarly, both "/" and "\" (syntactically denoting "missing something on the right," or "on the left," resp.) get interpreted semantically as a type-level version of *abstraction*. More precisely, if f is a unary function $f : A \to B$, then $f \in A\backslash B$ and $f \in B/A$.

The syntactic inference rules then also have a homomorphic image at the semantic level. The above two examples now become

$$\frac{B \bullet (B\backslash A)}{A}\, \beta_l$$

and

$$\frac{(A/B) \bullet B}{A}\, \beta_r$$

These inference rules are thus easily recognized as type-level versions of the β-reduction rule of the Lambda calculus: application cancels (left and right) abstraction.

8.3.4 The Lambek Calculus as a Type Logic

The semantic version of the Lambek Calculus can thus be understood as a type logic. The syntactic categories are interpreted as semantic types; the syntactic categorial operators $\bullet$, $\backslash$, and / are interpreted as type constructors, namely type-level versions of application and abstraction; and the syntactic categorial inference rules correspond to semantic type inference rules.

This is the interpretation that is commonly used as a basis of the modern application of the Lambek calculus in the formal semantics field of linguistics, as evidenced by the title of the survey on this area in the *Handbook of Logic and Language* [25], *Categorial Type Logics* [24], and it is this interpretation that we will apply to the content markup language OpenMath in the following.

8.4 Lambek Types and the Lambda Calculus

As a first step in our construction of a categorial type logic for OpenMath, we will briefly take a closer look at how the Lambek type calculus can be seen as a type-level extension of the Lambda Calculus.

Given the original interpretation of $\bullet$ as a type-level version of application, we will assign to Lambda Calculus expressions of the form fxy the type $F \bullet X \bullet Y$, where upper case letters denote the type of the OpenMath objects denoted by lower case letters.

Given this assignment, Lambda Calculus expressions like

$$((\lambda x.(\lambda y.fxy))x)y \equiv fxy$$

most naturally correspond to Lambek type expressions of the form

$$((X\backslash(Y\backslash(F \bullet X \bullet Y))) \bullet X) \bullet Y \equiv F \bullet X \bullet Y$$

Given the Lambda Calculus syntax ordering of these ingredients, we use a "left division" notation for lambda abstraction here.

Given this kind of type assignments, we then have type-level equivalents of Lambda Calculus reduction rules. β-Reduction, for example, which allows (for example)

$$\frac{(\lambda x.f)x}{f}\,\beta$$

corresponds to

$$\frac{(X\backslash F)\bullet X}{F}\,\beta_{lr}$$

In other words, (right) application cancels (left) abstraction just like multiplication cancels division.

Notice how the *left* abstraction notation $X\backslash F$ causes it to become visually separated from its canceling *right* application $F\bullet X$. We therefore propose to use in the following the much more intuitive though clearly equivalent *right* abstraction notation F/X instead of $X\backslash F$, which produces the following categorial version of the Lambda Calculus β-reduction rule:

$$\frac{(F/X)\bullet X}{F}\,\beta_{r}$$

In this notation, the Lambda calculus currying rules for binary functions

$$\lambda x.\lambda y.f \equiv \lambda x.(\lambda y.f)$$

(associativity rule for lambda abstraction) and

$$fxy \equiv (fx)y$$

(associativity rule for application) correspond to the following categorial type-level equivalences:

$$F/Y/X \equiv (F/Y)/X$$

$$F \bullet X \bullet Y \equiv (F \bullet X) \bullet Y$$

Notice how the switch to the more intuitive right abstraction division notation induces a reversal of the ordering of the variables (or rather, their types) in the type-level equivalent formula. This is a small price to pay for an otherwise nicely intuitive formalism which gives us, for example:

$$(((F/Y)/X) \bullet X) \bullet Y \equiv F \equiv (F/Y/X) \bullet X \bullet Y$$

Cancelling "factors" are now always directly adjacent.

This is therefore the variation on Lambek's notation for categorial type logics that we propose to use in the following for content markup languages, since these languages, like the Lambda Calculus that they tend to be based on, are essentially prefix notations that do not need to distinguish between right- and left-hand side arguments.[5]

8.5 Categorial Type of OpenMath Objects

We will now develop a categorial type assignment for OpenMath objects as defined in the OpenMath Standard Version 1.0 [10].[6] Following a compositional approach, we will examine each *structural syntactic* constructor of OpenMath and give it a *categorial* interpretation, i.e. propose a categorial type constructor as a homomorphic image to its syntactic "type".[7]

[5]This is actually closer to an older variant of this notation used by K. Ajdukiewicz in [147, 148] for a non-directional categorial calculus: $\frac{A}{B}$ instead of our A/B.

[6]Note that an update to the OpenMath Standard [10] document is currently being discussed in the OpenMath working group which includes problem fixes proposed in this section.

[7]We will use the notations used in the OpenMath Standard [10] to write OpenMath objects, and the notation in [24] for type assignment: if Ω denotes an OpenMath object, we let $\tau(\Omega)$ denote its type. More generally, we will let $\tau_\eta(\Omega)$ denote its type given η (the *environment*, a list of type assignments to variables).

In OpenMath 1.0, OpenMath objects are defined as abstract data structures that are either basic OpenMath objects or compound OpenMath objects. Compound OpenMath objects are either OpenMath application objects, OpenMath binding objects, OpenMath attribution objects, or OpenMath error objects; basic OpenMath objects are OpenMath symbols, OpenMath variables, OpenMath integers, OpenMath floating point numbers, OpenMath bytearrays, or OpenMath character strings.

Such OpenMath objects are also viewed as the result of parsing their *encodings*, i.e. as an abstract syntactic structure. It is in this interpretation that we use their definition as a basis for our categorial analysis in the following.

8.5.1 OpenMath Application Objects

Syntactically, an *OpenMath application object* has the form $\mathbf{application}(a_0, \ldots, a_n)$, where the a_i are arbitrary OpenMath objects and $n \geq 0$. a_0 is called the *head*,[8] the other a_i are the *arguments* of the OpenMath application object.

Given its intended meaning as *application*, we can define the categorial type of an OpenMath application object as:

$$\tau_\eta(\mathbf{application}(a_0, \ldots, a_n)) := (\tau_\eta(a_0) \bullet \ldots \bullet \tau_\eta(a_n))$$

This can also be expressed as the axiom schema

$$\frac{\tau_\eta(\mathbf{application}(a_0, \ldots, a_n))}{(\tau_\eta(a_0) \bullet \ldots \bullet \tau_\eta(a_n))}\, oma_n$$

This is essentially the same as definitions used in both the Small Type System [31] and Strong OpenMath [32] type system proposals for OpenMath, and it is rather straightforward.

Note that we have thus extended the categorial notation for type-level application to the case of nullary application, by writing $(\tau_\eta(a_0)\bullet)$ in the case of $n = 0$.

[8] The term *head* is used here based on our suggestion, which was made because of the parallel we perceived to its use in linguistic syntax theories.

8.5.2 OpenMath Attribution Objects

OpenMath attribution objects have the form $\mathbf{attribution}(a_1\ b_1, \ldots, a_n\ b_n, e)$. The *attribute names* a_i are restricted to be OpenMath symbols, whereas the *attribute values* b_i and the *body* e are arbitrary OpenMath objects.

Neither of the two existing proposals for OpenMath type systems deals with general OpenMath attribution objects. This is understandable since semantically, the "value" of an OpenMath attribution object is meant to be the embedded body, while the attributions are meant to provide additional information about the body OpenMath object without actually "changing" its meaning. Such attributes have been envisaged for expressing presentation hints for rendering the attributed body, or for explicitly providing type information about the attributed body OpenMath object. However, it is important to note that these usage guidelines are purely informal and implicit; the OpenMath Standard proper remains silent on these details.

As a categorial type definition for an OpenMath attribution object with a single attribute we propose

$$\tau(\mathbf{attribution}(a\ b, e)) := (\tau(a) \bullet \tau(b)) \bullet \tau(e)$$

or, in proof tree format, with the unchanged environment η made explicit,

$$\frac{\tau_\eta(\mathbf{attribution}(a\ b, e))}{(\tau_\eta(a) \bullet \tau_\eta(b)) \bullet \tau_\eta(e)}\, omattr_1$$

This corresponds to an interpretation of an OpenMath attribution object as applying the attribution, that is the result of applying the attribute to the attribute value, to the attributed OpenMath object:

$$\mathbf{attribution}(a\ b, e) \equiv (a(b))(e)$$

The OpenMath Standard declares that the following semantic equivalence holds for OpenMath attribution objects:

$$\textbf{attribution}(a_1\ b_1, a_2\ b_2, e) \equiv \textbf{attribution}(a_1\ b_1, \textbf{attribution}(a_2\ b_2, e))$$

from which we get the following fully general type assignment for OpenMath attribution objects:

$$\tau(\textbf{attribution}(a_1\ b_1, \ldots, a_n\ b_n, e)) := (\tau(a_1) \bullet \tau(b_1)) \bullet (\ldots((\tau(a_n) \bullet \tau(b_n)) \bullet \tau(e))\ldots)$$

or as a categorial type inference axiom schema

$$\frac{\tau_\eta(\textbf{attribution}(a_1\ b_1, \ldots, a_n\ b_n, e))}{(\tau_\eta(a_1) \bullet \tau_\eta(b_1)) \bullet (\ldots((\tau_\eta(a_n) \bullet \tau_\eta(b_n)) \bullet \tau_\eta(e))\ldots)}\ omattr_n$$

Note that this does not capture the restriction that annotating an OpenMath object shouldn't change its meaning; we will return to this question later in our discussion of signatures for attribute name symbols.

8.5.3 OpenMath Binding Objects and OpenMath Variables

An *OpenMath binding object* has the form $\textbf{binding}(b, v_1, \ldots, v_n, e)$. The components of a OpenMath binding object are its *head* b, called a *binder* if it is an OpenMath symbol, its *bound variables* $v_1 \ldots v_n$, and its *body* e. The binder and the body are both arbitrary OpenMath objects, and the bound variables are either OpenMath variables or OpenMath variables embedded as the body of an OpenMath attribution object.

Caprotti and Cohen [32] deal with OpenMath binding objects only in the case of a single, explicitly typed bound variable. Their proposal can be expressed in our categorial formalism roughly as

$$\tau(\textbf{binding}(b, \textbf{attribution}(\texttt{type}\ t, v), e)) := \tau(b) \bullet \tau(t) \bullet (\tau(e)/\tau(t))$$

Since we are aiming to be fully general, however, we need to define a categorial type for all OpenMath binding objects. We therefore offer the following simplified and generalized

compositional type assignment for OpenMath binding objects:[9]

$$\tau(\mathbf{binding}(b, v_1, \ldots, v_n, e)) := \tau(b) \bullet (\tau(e)/\tau(v_n)/\cdots/\tau(v_1))$$

This corresponds to an interpretation of an OpenMath binding object as forming an abstraction over the variables $v_1 \ldots v_n$ with body e and applying the binder b to the resulting abstraction: $\mathbf{binding}(b, v_1, \ldots, v_n, e) \equiv b(\lambda v_1. \cdots \lambda v_n.e)$.

As before for nullary application, OpenMath objects allow nullary abstraction ($n = 0$), for which we need to extend the existing notation slightly:

$$\tau(\mathbf{binding}(b, e)) := \tau(b) \bullet (\tau(e)/)$$

Environment

These type assignments have a nicely simple form, but they still leave implicit the fact that variable binding modifies the environment η of type assignments to variables.[10] The following axiom schema makes this explicit in the case of OpenMath variables $v_1 \ldots v_n$:

$$\frac{\tau_\eta(\mathbf{binding}(b, v_1, \ldots, v_n, e))}{\tau_\eta(b) \bullet (\tau_{[v_1:V_1,\ldots,v_n:V_n|\eta]}(e)/V_n/\cdots/V_1)}\, omb_{v_1:V_1,\ldots,v_n:V_n}$$

Attributed Bound Variables

This still ignores OpenMath binding objects in which the bound variable OpenMath objects are not simple OpenMath variables but attributed OpenMath variables.

[9]Note that we have lost the type information that was present in Caprotti and Cohen's proposal; by providing a general type semantic for OpenMath attribution objects and by giving appropriate signatures to type symbols we can recover this information in our framework.

[10]In $\lambda x.(\lambda x.\sin x)(x\pi)$, for example, the outer occurrence of x might reasonably expected to be of type *rational number*, while the inner occurrence would have to be of type *real number* in this case. Thus, we expect this expression to have a categorial type of $(((((\mathbb{R}/\mathbb{R}) \bullet X_2)/X_2) \bullet ((\mathbb{R}/\mathbb{Q}/\mathbb{R}) \bullet X_1 \bullet \mathbb{R}))/X_1)$, where without the extra complication of an explicit environment we might incorrectly have read the implicit rules to give $(((((\mathbb{R}/\mathbb{R}) \bullet X)/X) \bullet ((\mathbb{R}/\mathbb{Q}/\mathbb{R}) \bullet X \bullet \mathbb{R}))/X)$ with the incorrect reading that both x's necessarily have the same type. The environment also allows us to explicitly identify the type of the variable bound by a λ with the type of an occurrence of that variable in the body of that λ.

It would be nice if we could use the same type assignment as above in a slightly generalized form, but if we abbreviate $\eta' := [v_1 : V_1, \ldots, v_n : V_n \mid \eta]$ and denote the attributed variables av_k and the variables themselves as v_k, the rule[11]

$$\frac{\tau_\eta(\mathbf{binding}(b, av_1, \ldots, av_n, e))}{\tau_\eta(b) \bullet (\tau_{\eta'}(e)/\tau_\eta(av_n)/\cdots/\tau_\eta(av_1))}\, omb^*_{v_1:V_1,\ldots,v_n:V_n}$$

gives the wrong categorial types for the abstractions (because $\tau_\eta(v_k) \neq V_k$, the required categorial type of the bound variables – instead, $\tau_{\eta'}(v_k) = V_k$), and the rule

$$\frac{\tau_\eta(\mathbf{binding}(b, av_1, \ldots, av_n, e))}{\tau_\eta(b) \bullet (\tau_{\eta'}(e)/\tau_{\eta'}(av_n)/\cdots/\tau_{\eta'}(av_1))}\, omb^{**}_{v_1:V_1,\ldots,v_n:V_n}$$

puts the variables' attributes inside the variables' scope, which is not the intention of the OpenMath specification which states that the scope of the variables is the body e.

In the case of a single bound variable with a single attribution, what we would really like instead is a rule like this one, which gets both the scope of the attribution and the type of the bound variable right and at the same time is consistent with the categorial type assignment for an OpenMath attribution object that we proposed above:

$$\frac{\tau_\eta(\mathbf{binding}(b, \mathbf{attribution}(a_1\ b_1, v), e))}{\tau_\eta(b) \bullet (\tau_{[v:V|\eta]}(e)/((\tau_\eta(a_1) \bullet \tau_\eta(b_1)) \bullet V))}\, omb_{v:V}$$

Unfortunately, generalizing this formula to arbitrary numbers of variables and arbitrary numbers of attributions per variable yields a completely unwieldy formula despite being entirely straightforward.[12] We will therefore leave it as an exercise to the reader.

[11] Following linguistics literature traditions, we use asterisks * to mark faulty examples.

[12] Generalizing to multiple attributes for a single bound variable gives

$$\frac{\tau_\eta(\mathbf{binding}(b, \mathbf{attribution}(a_1\ b_1, \ldots, a_k\ b_k v), e))}{\tau_\eta(b) \bullet (\tau_{[v:V|\eta]}(e)/((\tau(a_1) \bullet \tau(b_1)) \bullet (\ldots((\tau(a_k) \bullet \tau(b_k)) \bullet V)\ldots))}\, omb_{v:V}$$

while the generalization to several bound variables looks as follows:

OpenMath Variables

For *OpenMath variables*, the categorial type would generally be X – a type variable:

$$\frac{\tau_{\eta_{x:X}}(x)}{X}\, omv_x$$

where $\eta_{x:X}$ stands for the type assignment X for the variable x in the environment η.

Thus, the environment η contains type assignments for variables that are themselves variables, namely, type variables. Any concrete type assignment for a variable in an OpenMath object needs to be deduced via type inference mechanisms and unification.

Care needs to be taken therefore that several occurrences of a variable in the same scope are properly assigned the same type variable as their lexical meaning assignment, and that the same variable name in different scopes is assigned different type-level variables in the type assignment. That is the task of the environment parameter η of the type assignment $\tau_\eta(\Omega)$.

Letting $\eta_{x:X}$ stand for the type assignment $(x : X) \in \eta$ such that if the list η contains the type assignments $(x : X_1), (x : X_2), \ldots$ in this order, then $X = X_1$ (i.e. letting the environment η act as a stack of type assignments for variables in the standard fashion), we can now see that the above definition for type assignments for variables, bindings, and attributions combine to give the expected results:

$$\cfrac{\cfrac{\cfrac{\cfrac{\tau_{[]}(\mathbf{binding}(a, v, \mathbf{application}(f, v, v)))}{\tau_{[]}(a) \bullet (\tau_{[v:V]}(\mathbf{application}(f, v, v))/\tau_{[v:V]}(v))}\, omb_{v:V}}{\tau_{[]}(a) \bullet (\tau_{[v:V]}(\mathbf{application}(f, v, v))/V)}\, omv_v}{\tau_{[]}(a) \bullet ((\tau_{[v:V]}(f) \bullet \tau_{[v:V]}(v) \bullet \tau_{[v:V]}(v))/V)}\, oma_2}{\tau(a)_{[]} \bullet ((\tau_{[v:V]}(f) \bullet V \bullet V)/V)}\, omv_v$$

$$\cfrac{\tau_\eta(\mathbf{binding}(b, \mathbf{attribution}(a_1\ b_1, v_1), \ldots, \mathbf{attribution}(a_n\ b_n, v_n), e))}{\tau_\eta(b) \bullet (\tau_{[v_n:V_n,\ldots,v_1:V_1|\eta]}(e)/((\tau_\eta(a_1) \bullet \tau_\eta(b_1)) \bullet V_1)) \ldots /((\tau_\eta(a_n) \bullet \tau_\eta(b_n)) \bullet V_n))}\, omb_{v_1:V_1,\ldots v_n:V_n}$$

This unwieldiness suggests that there is a problem here, either with the method of categorial type assignment we are proposing here or with the OpenMath language. We will discuss this question below in the context of more such problems.

but

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau_{[]}(\mathbf{binding}(a, v, \mathbf{application}(f, v, \mathbf{binding}(b, v, v)))}{\tau_{[]}(a) \bullet (\tau_{[v:V]}(\mathbf{application}(f, v, \mathbf{binding}(b, v, v)))/\tau_{[v:V]}(v))}\, omb_{v:V}}{\tau_{[]}(a) \bullet (\tau_{[v:V]}(\mathbf{application}(f, v, \mathbf{binding}(b, v, v)))/V)}\, omv_v}{\tau_{[]}(a) \bullet ((\tau_{[v:V]}(f) \bullet \tau_{[v:V]}(v) \bullet \tau_{[v:V]}(\mathbf{binding}(b, v, v)))/V)}\, oma_2}{\tau_{[]}(a) \bullet ((\tau_{[v:V]}(f) \bullet V \bullet \tau_{[v:V]}(\mathbf{binding}(b, v, v)))/V)}\, omv_v}{\tau_{[]}(a) \bullet ((\tau_{[v:V]}(f) \bullet V \bullet (\tau_{[v:V]}(b) \bullet (\tau_{[v:V_1, v:V]}(v)/\tau_{[v:V_1, v:V]}(v))))/V}\, omb_{v:V_1}}{\tau_{[]}(a) \bullet ((\tau_{[v:V]}(f) \bullet V \bullet (\tau_{[v:V]}(b) \bullet (V_1/V_1)))/V)}\, omv_v$$

Variations on a Theme

In the preceding, we used a standard technique described in [23] for constructing a compositional semantics for the fragment of OpenMath we have been dealing with so far. This technique uses a meta-level to make sure that type variables have the correct scopes in the categorial type expression.

In modern treatments of categorial type logics in a linguistics setting such as [24], this appears to be a standard way to treat linguistic binding and quantification phenomena, too.

Since the formalism we have used does get a bit unwieldy once we explicitly introduce the environment auxiliary, we should take a moment to consider a historical alternative proposed by Ajdukiewicz in [147, 148] in 1935 when he considered the application of his version of a categorial type logic to mathematical formulas in general, and mathematical operators that bind variables in particular.

Essentially, Ajdukiewicz proposed to add a piece of formalism to denote the barrier that a variable binding operator presents at the categorial type level. Here is an example from his paper representing the categorial type of $\forall x.p$:

$$| \frac{s}{s} s$$

The vertical bar denotes the barrier induced by the binding in his formalism.

In a more modern resurrection within the framework of the Lambek calculus, we might render this idea using a slight variation on this theme in the case of OpenMath binding objects:

$$\frac{\tau(\mathbf{binding}(b, v, e))}{\tau(b) \bullet [\tau(e)/V]}$$

Here, the square brackets denote Ajdukiewicz's barrier, a scope boundary for the type variable(s) abstracted in the barrier. With an appropriate type logic, this would allow us to write, for example,

$$\frac{\tau(\mathbf{binding}(b, v, \mathbf{binding}(b, v, e)))}{\tau(b) \bullet [\tau(b) \bullet [\tau(e)/V]/V]}$$

and to mean that the two occurrences of V do not denote the same type variable.

Such a formalism would allow us to leave off an explicit environment η, but on the other hand it is obvious that after simple α-conversion (variable renaming to remove name clashes), which is implemented by the environment, the square brackets are no longer really required, and we would be back with the same Lambek calculus that we have been using so far. Thus, at the cost of complicating the construction of the categorial type for an OpenMath object we have simplified the categorial type itself considerably (using two instead of three primitive constructors).

8.5.4 OpenMath Error Objects

OpenMath error objects, semantically speaking, only carry the meaning that an error occurred. For the sake of completeness, we therefore simple note that, in a type inference chain, their occurrence would therefore simply trigger a "fail" condition, or $\perp$:

$$\frac{\tau_\eta(\mathbf{error}(\ldots))}{\perp}\, omerr$$

8.5.5 OpenMath Symbols

Following Moortgat [24], and "in accordance with the categorial tenet of *radical lexicalism*, we assume that the grammar for [OpenMath] is given by the conjunction of [its] general type

logic with [its] language-specific lexicon LEX([OpenMath]) [which is] characterized in terms of a type assignment function" [24] that assigns a set of categorial types to a lexical item.[13]

In the case of *OpenMath symbols*, OpenMath realizes LEX in the form of OpenMath content dictionaries and zero or more associated "Signature" declarations. Signature declarations depend on the type system they are embedded in.[14]

We will discuss in detail some common patterns of categorial signatures of OpenMath symbols below.

8.5.6 Other Basic OpenMath Objects

OpenMath distinguishes several different kinds of leaves, or *basic OpenMath objects*. Since these are "syntactically" distinguished,[15] we need to discuss separate categorial type assignment rules for each kind of basic OpenMath objects.

In the interest of a "purely" categorial semantics, we could assign all the other constant basic OpenMath objects the type X – i.e., unknown, variable. However, OpenMath integers are one example where the intent is quite clearly that they be representations of integers, and their type thus $\mathbb{Z}$. Both kinds of strategy are sanctioned for a categorial OpenMath grammar that we quoted above.

Both strategies can indeed be found in the two existing OpenMath type system proposals. Davenport [31] introduces "meaningful variable names" as type designators in his signatures, which is formally equivalent to the first alternative of assigning all basic OpenMath objects a type variable as type. Caprotti and Cohen [32] introduce special type name constants for all basic OpenMath object types, but OpenMath integers do *not* have type $\mathbb{Z}$ in their system.

[13] The type assigned to a lexical item therefore does not need to be unique, though it is assumed that there are only a finite number of distinct alternatives. [24]

[14] At least two such systems have already been defined for OpenMath, but none cover all OpenMath content dictionarys.

[15] OpenMath objects are actually declared as an abstract data type that can have several different syntactic variants in actual OpenMath Encodings." We consider them as abstract syntax trees here, which renders the categorial type logics tools applicable in this case.

8.6 Categorial Type Inference Rules and OpenMath

Now that we have defined the basic translations for OpenMath object structures into categorial types, we need to choose which categorial type inference rules should apply to them.

8.6.1 β-Reduction

Clearly, the type-level version of β-reduction needs to apply to OpenMath types if we want to capture the intended meanings of OpenMath application objects and OpenMath binding objects. Here is the corresponding categorial type inference rule schema:

$$\frac{(A/B_1/B_2\cdots/B_n)\bullet B_n\cdots\bullet B_2\bullet B_1}{A}\,om\beta_n$$

This simply means that applying an n-ary function object with argument types $B_1 \ldots B_n$ to n arguments of the correct type in the correct order, this reduces to an element of the result type of the function object.

Note that in linguistic applications, we only find a single inference rule of this type for $n = 1$. The reason for this is that in the Lambda calculus (which is used in linguistic applications) both application and abstraction obey an associativity law. OpenMath as such does *not* obey these laws (see below), so that we have to use a rule schema for the type-level version of β-reduction for all $n \geq 0$.

8.6.2 Currying

OpenMath, as it stands, provides very limited support for another type inference rule that is usually found in such contexts, namely the currying rules described previously, repeated here in the case of two arguments:

$$\frac{(A/B/C)}{((A/B)/C)}\,/assoc^*$$

and

$$\frac{A \bullet B \bullet C}{(A \bullet B) \bullet C} \bullet assoc^*$$

Neither of these rules is defined to hold for OpenMath objects in full generality.

The only currying rule that the OpenMath Standard in its version 1.0 clearly specifies holds for OpenMath binding objects and can be expressed as a categorial type inference rule as follows:

$$\frac{D \bullet (A/B/C)}{D \bullet ((D \bullet (A/B))/C)} om/assoc$$

This is modeled on the abstraction currying rule above which we can try to recover from the OpenMath binding object currying rule by letting[16] $D = (X/X)$ and applying the β-reduction rule for OpenMath.[17]

OpenMath application objects, on the other hand, do not come with a currying rule in the OpenMath Standard 1.0 – the symmetry between application and abstraction currying that we see in the usual definition above is broken here. Since the currying rules are actually derived from the observation that $(x, y \mapsto f)(a, b) = ((x \mapsto (y \mapsto f))(a))(b)$, a rule that applies only if both application *and* abstraction are curried, we have reason to be worried.

[16]Note the use of X to denote a second-order type variable.

[17]

$$\frac{(X/X) \bullet (A/B/C)}{A/B/C} om\beta_1,\ X = A/B/C$$

and

$$\frac{\dfrac{(X/X) \bullet (((Y/Y) \bullet (A/B))/C)}{(X/X) \bullet ((A/B)/C)} om\beta_1,\ Y = A/B}{(A/B)/C} om\beta_1,\ X = (A/B)/C$$

Note that we have to cheat a bit in the second derivation in order to achieve this result, by giving each copy of D a different copy of its signature (X/X vs. Y/Y).

We thus perceive a mismatch between the categorial type system we sketched above on the one hand, and the definitions that the OpenMath Standard 1.0 provides. This may be due to problems in the categorial semantics approach that we are advocating here, or it may be due to problems in the OpenMath definition. We argue here that there are problems in the OpenMath definition that our approach has flushed out.

The first problem that we found[18] can be appreciated by working out in detail the recovery of the standard formulation of abstraction currying from the type level OpenMath version: D in that equivalence unifies with $(A/B/C)/(A/B/C)$ and $((A/B)/C)/((A/B)/C)$ – and with $(A/B)/(A/B)$. The first pair is as we would expect, and the type inference mechanism should be able to handle it. The third variant however destroys any possibility of getting a consistent unification[19] for D: while currying does say that a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ may be interpreted as a function $f : \mathbb{R} \to (\mathbb{R} \to \mathbb{R})$, it is not true in general that it may also be interpreted as $f : \mathbb{R} \to \mathbb{R}$.

Another problem is that this rule makes it impossible to provide an OpenMath *formal mathematical property* definition of a binary function such as "`choose`," say, by specifying an equation of the form $\texttt{choose} := \lambda n, m.\frac{n!}{m!(n-m)!}$, for example.[20] Since OpenMath does not allow application currying, the binary function `choose` may only be called with two arguments. The formal mathematical property, however, given as an OpenMath object, licenses the equivalent formulation $\texttt{choose} := \lambda n.(\lambda m.\frac{n!}{m!(n-m)!})$, which accepts a single argument – an interpretation which is not licensed by OpenMath.

In a nutshell, breaking the application/abstraction symmetry with respect to associativity at the type level has some problematic, or at least annoying, formal consequences. On the other hand, had OpenMath specified a pair of general currying rules as at the start of this section, one might have hoped that these problems could have been avoided.

[18] Actually, we found a closely related one, namely, that the binder OpenMath object is both in and out of scope for the first bound variable if there is more than one variable bound in the OpenMath binding object, given the definition in the OpenMath Standard.

[19] This is why we had to "cheat a bit" to make the detailed derivation work in an earlier footnote.

[20] The syntax $\lambda n, m.f$ with multiple arguments to a lambda abstraction is not standard Lambda Calculus notation, as R. van Engelen pointed out to me (*personal communication*). It is used here in correspondence to the way that this formula would be represented in OpenMath, namely as an OpenMath binding object with head $\texttt{lambda}_{\texttt{fns1}}$, multiple bound variables, and the function body f as body. Strictly speaking, this should be rendered as $\lambda n.\lambda m.f$ to conform to the Lambda Calculus formalism.

However, the particular way that OpenMath binding objects are structured is already asymmetric with respect to the way that OpenMath application objects are structured, and this asymmetry dooms any attempt to introduce a general-purpose currying rule for OpenMath application objects and OpenMath binding objects, as we discovered.

The problem is the head of the OpenMath binding object, the binder. Some binders, such as universal, existential, and other quantifiers, and potential binders such as the sum, product, and other operators induced by associative n-ary functions, all obey the currying rule given in the OpenMath Standard. Unfortunately, there are other potential binders that do not curry at all.

The easiest way to see that is to analyze the type-level version of the OpenMath binding object currying rule as given earlier:

$$\frac{D \bullet (A/B/C)}{D \bullet ((D \bullet (A/B))/C)}\, om/assoc$$

Notice that the $(D \bullet (A/B))$ in the second line of this rule corresponds to the A in the first line. What this means in practical terms is that an OpenMath binding object needs to return the same type of result as its body returns in order for this currying rule to even make sense.

Quantifiers have bodies that return a truth-value, and they return a truth-value themselves; operators induced by n-ary associative functions such as addition, multiplication, union, intersection and many others return the same type of value as their bodies do. In addition, the currying rule is an axiom of the Lambda calculus – all the binder symbols defined in early OpenMath versions in fact obeyed the currying rule.

Unfortunately, not all possible binders do fit this mold. Here are a few well-known counter-examples:

***the*:** Sometimes written as ι, this binder would be used to represent something like "*the unique* x such that $P(x)$ is true": $\iota x.P(x)$. The body P returns a truth value, the generalized quantifier ι returns an object from an arbitrary domain.

***any*:** Chooses "*any* x that satisfies $P(x)$": $\epsilon x.P(x)$.

***sequence*:** $(A_i)_{i\in\mathbb{N}}$ – the *sequence* of all A_i where i ranges over the natural numbers, for example. The body's return type is the type of the *elements* of the sequence, but the return type of the generalized quantifier is the type of the sequence itself.

***set of*:** $\{x \mid P(x)\}$ – the *set of* all x that satisfy $P(x)$.

Clearly, the class of counter-examples is open-ended, contains very important concepts, and is therefore anything but negligible. The currying rule for OpenMath binding objects is irreparably broken.

An update for OpenMath version 1 is therefore now being circulated for discussion that removes the currying rule completely, as one of two main bug fixes.[21]

8.7 Lexical Semantics of OpenMath

Recall that "in accordance with the categorial tenet of *radical lexicalism*, we assume that the grammar for [OpenMath] is given by the conjunction of [its] general type logic with [its] language-specific lexicon LEX([OpenMath]) [which is] characterized in terms of a type assignment function" [24] that assigns a set of categorial types to a lexical item.

While the detailed lexical semantics of the *full* set of OpenMath symbols, i.e. the full definition of LEX(OpenMath), is explicitly not within the scope of this chapter – the whole point of a categorial semantics is the ability to factor out that aspect of meaning – it *is* of interest here to study what *kinds* of signatures an OpenMath categorial type logic might have to support, i.e. what entries in LEX(OpenMath) might look like, and if the mechanisms we have introduced so far are up to the task of handling them.

8.7.1 Constants, Functions, and Predicates

Regular, fixed-arity functions or relations would typically be assigned signatures of the form $\tau(\mathtt{pi}_{\mathtt{nums1}}) := \mathbb{R}$ or $\tau(\mathtt{sin}_{\mathtt{transc1}}) \in \{(\mathbb{R}/\mathbb{R}), (\mathbb{C}/\mathbb{C})\}$, or as type inference rules,

$$\frac{\tau(\mathtt{pi}_{\mathtt{nums1}})}{\mathbb{R}}\ \text{LEX}$$

[21] The results reported here were directly responsible for both discovering and fixing this major bug in OpenMath 1.0 [10].

and

$$\frac{\tau(\mathtt{sin}_{\mathtt{transc1}})}{\mathbb{C}/\mathbb{C}}\,\mathrm{LEX} \qquad\qquad \frac{\tau(\mathtt{sin}_{\mathtt{transc1}})}{\mathbb{R}/\mathbb{R}}\,\mathrm{LEX}$$

Note that the categorial types approach can assign more than one signature to a symbol, though only a finite number are typically allowed [24].[22]

Note also that we use concrete mathematical type constants such as $\mathbb{R}$ for the real numbers and $\mathbb{C}$ for the complex numbers in these examples instead of the type variables that we have used in our examples so far. This is in keeping with the OpenMath philosophy that signatures of OpenMath Symbols are given modulo a concrete type system.

Predicates work similarly to functions, generally, e.g.

$$\frac{\tau(\mathtt{geq}_{\mathtt{relation1}})}{(2/\mathbb{R}/\mathbb{R})}\,\mathrm{LEX}$$

with "2" standing for the set of truth values.

8.7.2 Interaction of Categorial and Lexical Type Systems

Here is a simple example of how the categorial and a concrete lexical type system might interact profitably to deduce the type of $\sin\pi$:

$$\cfrac{\cfrac{\tau(\mathbf{application}(\mathtt{sin}_{\mathtt{transc1}}, \mathtt{pi}_{\mathtt{nums1}}))}{(\tau(\mathtt{sin}_{\mathtt{transc1}}) \bullet \tau(\mathtt{pi}_{\mathtt{nums1}}))}\,\mathrm{LEX}}{\cfrac{((\mathbb{R}/\mathbb{R}) \bullet \mathbb{R})}{\mathbb{R}}\,\beta}\,oma$$

We can thus infer that $\sin\pi \in \mathbb{R}$.

[22] The Small Type System for OpenMath only allows a single signature per OpenMath symbol. For more complex type systems this restriction is not reasonable. In the example of the sine function, the second signature for sine as a real function is necessary to declare that the sine of real numbers is a real number to which other real functions or predicates can thus be applied, for example.

On the other hand, for $\pi(\sin)$ we get

$$\dfrac{\dfrac{\tau(\mathbf{application}(\mathtt{pi}_{\mathtt{transc1}}, \mathtt{sin}_{\mathtt{transc1}}))}{(\tau(\mathtt{pi}_{\mathtt{transc1}}) \bullet \tau(\mathtt{sin}_{\mathtt{transc1}}))}\ \mathrm{LEX}}{(\mathbb{R} \bullet (\mathbb{R}/\mathbb{R}))}\ oma$$

which does not reduce any further.

Thus, our examples will use concrete types "plugged into" the categorial type system from a concrete type system for illustration purposes. This shows the usefulness, but also the limitations of the concept of a categorial type system: it mirrors very well the partitioning of the OpenMath language into the fixed definition of the structure of OpenMath objects on the one hand and the open and extensible set of OpenMath symbol definitions in OpenMath content dictionaries on the other, dealing only with the former, but leaving hooks for the latter.

The important difference between this approach and previous type system proposals for OpenMath is that in our example above the signatures of symbols do not consist solely of constructors and constants of a separate type theory, but utilize the standardized categorial type constructors for application and abstraction in conjunction with constants from a sample concrete type system.

This corresponds to a proposal for OpenMath Signatures to utilize a standardized set of type constructor names for the type-level abstraction and application operators, as a least common denominator of admissible type systems for OpenMath. Different concrete type systems for OpenMath would then be seen as "plugging into" the categorial type system that we propose here as an extension to the actual OpenMath standard itself.

Simple and elegant though this idea may sound in theory, it immediately brings up the practical question of how a categorial type theory like the one we propose, on the one hand, and a concrete "plug-in" lexical type theory, on the other, would interact mathematically and proof-theoretically.

To see that this question is not quite trivial, consider the deceptively simple example formula $(\sin \pi = 0)$ expressed in OpenMath. A few special predicates such as equality tend to be polymorphic, applying to any types at all. We can express this easily in the categorial

framework by using type variables instead of constant type names in the signature for the equality predicate:

$$\frac{\tau(\mathtt{eq}_{\mathtt{relations1}})}{2/X/X}\,\text{LEX}$$

Given this, here is then a derivation of the fact that $(\sin\pi = 0)$ stands for a truth value:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\tau(\mathbf{application}(\mathtt{eq}_{\mathtt{relations1}}, \mathbf{application}(\mathtt{sin}_{\mathtt{transc1}}, \mathtt{pi}_{\mathtt{nums1}}), 0))}{\tau(\mathtt{eq}_{\mathtt{relations1}}) \bullet (\tau(\mathtt{sin}_{\mathtt{transc1}}) \bullet \tau(\mathtt{pi}_{\mathtt{nums1}})) \bullet \tau(0)}\,oma}{(2/X/X) \bullet ((\mathbb{R}/\mathbb{R}) \bullet \mathbb{R}) \bullet \mathbb{Z}}\,\text{LEX}}{(2/X/X) \bullet \mathbb{R} \bullet \mathbb{Z}}\,\beta_1}{(2/X/X) \bullet \mathbb{R} \bullet \mathbb{R}}\,\mathbb{Z} \subset \mathbb{R}}{2}\,\beta_2(X = \mathbb{R})$$

This derivation uses two important new details. First, as a new aspect of the categorial type logic, it uses unification to derive an appropriate value for the type variable X from the structure of the β-reduction rule. Second, it includes a derivation step that depends exclusively on the plugged-in concrete lexical type theory ($\mathbb{Z} \subset \mathbb{R}$), in this case, a simple deduction based on a simple type hierarchy.

Together, these make the question of an interaction between a categorial type logic and a "plug-in" lexical type theory quite non-trivial. Luckily, this question has been studied extensively and in a very general form in the categorial semantics research area of formal linguistics, from where we "stole" the idea in the first place – another good reason for exploring the linguistic parallel in the study of content markup languages. In [24] we can see that the categorial type logic that we have proposed for OpenMath is a simple version of the **L2** type logic, the positive features of which it thus inherits. In addition, [151] "explores the consequences of layering a Lambek calculus over an arbitrary constraint logic," and "[provides] for [such a] hybrid language a simple model-theoretic semantics, [where] the proof system for the underlying base logic" (in our context, the type logic for a concrete "plug-in" type system) "can be assumed to be a black box that performs *entailment checking*" (such as subtype checking for the plugin type system).

8.7.3 N-ary Functions and Relations

So far we have seen signature entries for simple functions, predicates, and constants, all of which easily fit into the categorial type system we described so far. However, OpenMath has the concept of n-ary functions, which corresponds to that of a family of signatures:

$$\tau(\mathtt{plus}_{\mathtt{arith1}}) \in \{(\mathbb{C}/\mathbb{C}), \ldots, (\mathbb{C}/\cdots/\mathbb{C}), \ldots\}$$

which we might conveniently abbreviate by introducing the notation $/^{\omega}$ for n-ary abstraction:

$$\frac{\tau(\mathtt{plus}_{\mathtt{arith1}})}{(\mathbb{N}/^{\omega}\mathbb{N})}\,\mathrm{LEX} \qquad \cdots \qquad \frac{\tau(\mathtt{plus}_{\mathtt{arith1}})}{(\mathbb{C}/^{\omega}\mathbb{C})}\,\mathrm{LEX}$$

The correct choice of a signature for the OpenMath n-ary function symbol depends on the context it is found in: the number of arguments it is called with determines its arity. Thus, we need to add the following reduction rule which defines n-ary abstraction in terms of (regular) abstraction and application:

$$\frac{(A/^{\omega}B) \bullet B_1 \cdots \bullet B_n}{(A\underbrace{/B\cdots/B}_{n\mathrm{times}}) \bullet B_1 \cdots \bullet B_n}\,om/^{\omega}{}_n$$

Note that the existence of such n-ary function symbols in OpenMath could cause complications if we had a general currying rule for the categorial semantics of OpenMath: n-ary functions cannot be curried.

Thus, we have here a first place where we really need to go beyond the regular set of ingredients of a categorial type system. However, as defined above, n-ary abstraction is little more than syntactic sugar since it reduces immediately via above $om/^{\omega}$-reduction rule to the ingredients of the regular categorial type system.

This is true as long as n-ary functions are only found applied to arguments, in which case this rule makes the n-ary abstraction from their signatures disappear. I will not analyze in detail what the effect of adding this ingredient has on the type analysis of OpenMath objects that include n-ary function symbols without this context, as for example in "$(\mathbb{R}, 0, 1, +, \times)$

is a field," but will only note that the problem we see here is very similar to the one that prompted LISP language designers to provide additional regular binary function versions for all (special) n-ary functions. It may well be a good idea for OpenMath content dictionary designers to follow the LISP example; in this case, the use of n-ary function symbols outside the head position of an OpenMath application object could be declared erroneous: liable to produce errors.

Thus, signatures for n-ary functions are not handled easily in the categorial type logic, but since they are not a concept of the OpenMath core, this problem is not as serious as it might seem.

8.7.4 Binder Symbols

Recall that the categorial type for an OpenMath binding object was defined as

$$\frac{\tau_\eta(\mathbf{binding}(b, v_1, \ldots, v_n, e))}{\tau_\eta(b) \bullet (\tau_{[v_1:V_1,\ldots,v_n:V_n|\eta]}(e)/V_n/\cdots/V_1)}\, omb_{v_1:V_1,\ldots,v_n:V_n}$$

For binder symbols, i.e. symbols that are meant to appear as the head b of a OpenMath binding object, we can use this categorial type inference rule in reverse,[23] as it were, to deduce from the type of a OpenMath binding object as given above that an OpenMath binder symbol needs to have a signature of the form $Y/(Z/X_n/\cdots/X_1)$ – returning objects of type Y given a mapping as an argument that returns an object of type Z if given objects of type $X_1, \ldots, X_n$. Quantifiers are examples that have this general form, having a signature $2/(2/X_n/\cdots/X_1)$, i.e. taking a predicate in n variables as an argument and producing a truth value.

To illustrate the range of possibilities given by this approach, here are some more examples of signatures of generalized quantifiers that have not been defined in OpenMath content dictionaries so far.

[23] This was not the most general form of the categorial type inference rule for OpenMath binding objects, since we ignore the potential attributions of the bound variable components of an OpenMath binding object, but since the categorial type inference rule for attributions is designed in such a way that the full categorial type of an OpenMath binding object with attributed variables reduces to the categorial type of an OpenMath binding object of the simpler form used here, we can use the simpler form without loss of generality.

***the*:** *The* is a generalized quantifier used in the definition of implicit functions, as in "*the* x such that $P(x)$ is true."

$$\frac{\tau(\texttt{the})}{X/(2/X)}\,\text{LEX}$$

***exists uniquely*:** *Exists uniquely* is a regular quantifier, sometimes rendered as $\exists!$ or as $\dot{\exists}$. It is used in formalizations of statements like "a field has a unique zero element."

$$\frac{\tau(\texttt{exists\_uniquely})}{2/(2/X_1/\ldots/X_n)}\,\text{LEX}$$

***set of all*:** A set constructor along the lines of "the *set of all* points (x, y, z) such that $x^2 + y^2 + z^2 < r^2$." Using $\mathfrak{P}(X)$ to denote the power-set of a set X, we can define the categorial type of this generalized quantifier as:

$$\frac{\tau(\texttt{set\_of\_all})}{\mathfrak{P}(Y)/(Y/X_1/\cdots/X_n)/(2/X_1/\cdots/X_n)}\,\text{LEX}$$

The *set of all* operator thus takes two functional arguments, namely the predicate that filters variable assignment combinations, and the function that constructs the elements of the set from the filtered combinations of values of the bound variables.

The first and third of these examples illustrates that there are genuine generalizations of quantifiers that are not easily reduced to "real" quantifiers, justifying our use of the term "generalized quantifiers" in this discussion. The second example shows that there are also frequently used "regular" quantifiers that go beyond the usual set of two, existential and universal.

The third example, finally, shows a serious limitation of OpenMath binding objects. The *set of all* operator cannot be represented as a binder, i.e. the head of an OpenMath binding object, because it requires two body abstractions, not just one. OpenMath currently only allows a single body in an OpenMath binding object.

However, the sample signature for the *set of all* symbol is still perfectly fine. Its *only* problem is that it cannot be used as the head of an OpenMath binding object; it still works just fine as the head of an OpenMath application object in conjunction with λ abstractions, e.g.

$$\mathbf{application}(\texttt{set\_of\_all}, \mathbf{binding}(\texttt{lambda}, x, x), \mathbf{binding}(\texttt{lambda}, x, \mathbf{application}(\texttt{gt}, x, 0)))$$

This observation leads us to make another proposal for future versions of OpenMath content dictionaries as a consequence of our proposal for a standardized categorial type system for OpenMath itself. We propose that any OpenMath object with the right signature be allowed as a OpenMath binding object header, and any binder be allowed as the head of OpenMath application object. As an example, the following two should both be allowed, and in applications with the necessary capabilities they should mean the same thing, namely $\forall x.x = x$:

$$\mathbf{binding}(\texttt{forall}_{\texttt{quant1}}, x, \mathbf{application}(\texttt{eq}_{\texttt{relations1}}, x, x))$$

and

$$\mathbf{application}(\texttt{forall}_{\texttt{quant1}}, \mathbf{binding}(\texttt{lambda}_{\texttt{fns1}}, x, \mathbf{application}(\texttt{eq}_{\texttt{relations1}}, x, x)))$$

This proposal also allows us to make sense of compound OpenMath objects as heads of OpenMath binding objects. Let us take the *exists uniquely* quantifier that we used as an example earlier, and which can be defined formally as

$$\dot{\exists} x.P(x) :\Leftrightarrow (\exists x.P(x) \wedge (\forall y.P(y) \Rightarrow y = x))$$

We can therefore define the unique-existential quantifier as an operator taking unary predicates as arguments, as follows:

$$\dot{\exists} := \lambda P.(\exists x.P(x) \wedge (\forall y.P(y) \Rightarrow y = x))$$

In OpenMath, the right-hand side of this definition is represented by an OpenMath object that begins as follows:

$$\mathbf{binding}(\texttt{lambda}, P, \mathbf{binding}(\texttt{exists}, x, \mathbf{application}(\texttt{and}, \mathbf{application}(P, x), ...)))$$

This OpenMath object can meaningfully appear as a binder, i.e. as the head of an OpenMath binding object, with the same meaning as a dedicated OpenMath symbol for the unique-existential quantifier (which did not exist yet in the first release of the OpenMath content dictionaries) would have.

Note that unlike the signature examples given earlier for function and constant symbols, which involved mostly only constant type names, the signatures for binders all involve type variables. These variables are free; a type inference mechanism would need to take this into account and perform proper variable renaming in a concrete implementation.

Note also that we do not quantify over these type variables; this makes it possible to keep the resulting second-order type system decidable [24].

8.7.5 Attribute Names

As in the case of binder symbol signatures, we can derive the typical form of the signature of an attribute name symbol from the categorial type inference rule for OpenMath attribution objects: $\tau(\mathit{attribute\_name}) = ((Y/X)/Z)$, where X is the type of the body, Y is the result type of the OpenMath attribution object, and Z is the type of the value of the attribute. In fact, since attributions are not supposed to change the meaning of the attributed OpenMath object, we can derive a slightly less general form of the signature of an attribute name symbol, $\tau(\mathit{attribute\_name}) = ((X/X)/Z)$ (i.e. the type of the body object is the same as that of the result type of the attribution), to capture this requirement[24] for the semantics of attributions that we had to leave out in our definition of the type of an OpenMath attribution object earlier:

$$\frac{\tau_\eta(\mathbf{attribution}(a\ b, e))}{(\tau_\eta(a) \bullet \tau_\eta(b)) \bullet \tau_\eta(e)}\ \mathit{omattr}_1$$

OpenMath attribution objects and their proper use have long been subjects of heated discussions, with little actual consensus to emerge. However, there are two particular classes of usage for such OpenMath objects that have been long and widely acknowledged to be a

[24]Note that this captures as well as possible the informal requirement that attributions do not change the semantics of their body objects, because clearly the attributed object and the unattributed one are different OpenMath objects, whose formal semantics can therefore only be "the same" in the sense of an equivalence relation, but not in the sense of a strict identity.

good rationale for including them: adding type information and adding rendering hints to an OpenMath object.

Needless to say, we aim to pursue a lexicalist agenda for attribute names, i.e. for the component of an OpenMath attribution object that corresponds to function or predicate names in OpenMath application objects and to binder names in OpenMath binding objects.

Here are a sample signature for an attribute name symbol that marks a presentation hint for the attributed OpenMath object:

$$\frac{\tau(\texttt{MathMLPresentation})}{(X/X)/\text{Mml-P}}\ \text{LEX}$$

This example illustrates how attribution does not usually change the type of the body OpenMath object while the type checking mechanism can still verify whether the attribute value is well-formed (in this case of type "Mml-P", which we may assume to stand for MathML Presentation).

8.7.6 Type Attribute Names

The signature for an OpenMath symbol used as an attribute name that marks an explicit type assignment for an OpenMath object is a more complicated case because such type assignments are meant to interact with concrete type logics that extend the underlying categorial type system. Caprotti and Cohen's treatment of explicitly typed OpenMath objects [32] therefore needed to give special treatment to this case.

The problem is that the attribute value associated with a $\texttt{type}_t$ attribute is an OpenMath object that *describes* a type in a particular type logic t. Examples for such attribute values are OpenMath symbols such as the OpenMath content dictionary `setname1`'s symbols for the sets $\mathbb{Z}$, $\mathbb{R}$, or $\mathbb{C}$, but also compound OpenMath objects that describe types like power-sets (e.g. $\mathfrak{P}(\mathbb{N})$) or categorial types (e.g. $(2/\mathbb{R}) \bullet \mathbb{R}$).

In order for such explicit type assignments to be available to a type inference mechanism, they first need to be *interpreted* as the (possibly compound) *type description* $\overline{T}_t$ of a type T in a type theory t. This is captured in the following generic signature for the attribute

name `type` in an OpenMath content dictionary t that contains the OpenMath symbols that denote the concepts of a concrete type theory t:

$$\frac{\tau(\texttt{type}_t)}{(T/T)/\overline{T}_t}\,\text{LEX}$$

This signature for a $\texttt{type}_t$ attribute symbol says that the type of the body OpenMath object and that of the resulting attributed OpenMath object must unify ("(T/T)"), true to the requirement that type attributes only *add to* but do not *change* the attributed OpenMath object. At the same time, however, the signature specifies that these types must unify with the type which is described by the attribute value of the $\texttt{type}_t$ attribute ("$(\ldots)/\overline{T}$").

Thus, we clearly need to treat type attributions of this kind specially if we want to handle explicitly typed OpenMath objects within the framework of the categorial type system for OpenMath. The need to introduce the new concept of a type description as something to be interpreted by the categorial type system underlies the need for introducing the new $\overline{T}_t$ notation for *description of type T in type theory t* into the set of fundamental ingredients of the categorial type system.[25]

8.7.7 Type Descriptor Symbols

From the preceding discussion, it is easy to see that the signature of type descriptor symbols that are to be used in OpenMath objects that are attribute values of a $\texttt{type}_t$ attribute need to involve the *type description* notion and notation.

As a simple example, let us consider the case of a type descriptor for the field of real numbers $\mathbb{R}$. When used as the name of a set in a regular mathematical formula, the type of this set might be given as "power-set of $\mathbb{R}$":

$$\frac{\tau(\mathtt{R}_{\texttt{setname1}})}{\mathfrak{P}(\mathbb{R})}\,\text{LEX}$$

[25] As far as I know, there is no equivalent in any human language that would correspond to such an explicit typing capability. Thus, the powerful analogy to formal linguistics that has guided our research into OpenMath and similar languages, up to and including the introduction of the concept of a categorial type system for OpenMath that underlies this very chapter, breaks down in an irremediable manner at this point.

However, when used in a type descriptor, the symbol $\mathtt{R}_t$ actually describes $\mathbb{R}$ itself, not its power-set, in the type theory t. Therefore, we get the following type assignment for $\mathtt{R}_t$ in its type-descriptor interpretation:[26]

$$\frac{\tau(\mathtt{R}_t)}{\overline{\mathbb{R}}_t}\ \text{LEX}$$

We can see that this is the right choice for the lexical type for this OpenMath symbol by testing it in the type analysis of an example, an OpenMath object that represents the concept of *real variable* x:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\tau_\eta(\mathbf{attribution}(\mathtt{type}_t\ \mathtt{R}_t, x))}{(\tau_\eta(\mathtt{type}_t) \bullet \tau_\eta(\mathtt{R}_t)) \bullet \tau_\eta(x)}\ omattr_1}{(((T/T)/\overline{T}_t) \bullet \overline{\mathbb{R}}_t) \bullet \tau_\eta(x)}\ \text{LEX}}{(((T/T)/\overline{T}_t) \bullet \overline{\mathbb{R}}_t) \bullet X}\ omv_x}{(\mathbb{R}/\mathbb{R}) \bullet X}\ \beta_1; \overline{T}_t = \overline{\mathbb{R}}_t}{\mathbb{R}}\ \beta_1; X = \mathbb{R}$$

Thus we get the correct type derivation of $\mathbb{R}$ for the typed variable x, as well as the unification of its type assignment X with $\mathbb{R}$ in the environment η.

Signatures for names of type constructors similarly involve mention of the type description notion. Here is an example:

$$\frac{\tau(\mathtt{abstraction}_t)}{(\overline{A_1/\cdots/A_n}_t)/\overline{A_1}_t/\cdots/\overline{A_n}_t}\ \text{LEX}$$

In words, the *abstraction* type constructor symbol stands for an operator that returns the description of a categorial abstraction $A_1/\cdots/A_n$ if we give it descriptions for types A_1 through A_n as arguments.

Again, we can test this against the example of a type attribution to a variable, for example declaring f to denote a unary real function:

[26]Note that it would be equally valid to introduce a new symbol for $\mathbb{R}$ in a new OpenMath content dictionary named, say, `typedesc`, that includes OpenMath symbols for generic type descriptors.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau_\eta(\mathbf{attribution}(\mathtt{type}_t\ \mathbf{application}(\mathtt{abstraction}_t, \mathtt{R}_t, \mathtt{R}_t), f))}{(\tau_\eta(\mathtt{type}_t) \bullet \tau_\eta(\mathbf{application}(\mathtt{abstraction}_t, \mathtt{R}_t, \mathtt{R}_t))) \bullet \tau_\eta(f)}\, omattr_1}{(\tau_\eta(\mathtt{type}_t) \bullet (\tau_\eta(\mathtt{abstraction}_t) \bullet \tau_\eta(\mathtt{R}_t) \bullet \tau_\eta(\mathtt{R}_t))) \bullet \tau_\eta(f)}\, oma_2}{(((T/T)/\overline{T}_t) \bullet (((\overline{A_1/A_2}_t)/\overline{A_1}_t/\overline{A_2}_t) \bullet \overline{\mathbb{R}}_t \bullet \overline{\mathbb{R}}_t)) \bullet \tau_\eta(f)}\, \text{LEX}}{(((T/T)/\overline{T}_t) \bullet (((\overline{A_1/A_2}_t)/\overline{A_1}_t/\overline{A_2}_t) \bullet \overline{\mathbb{R}}_t \bullet \overline{\mathbb{R}}_t)) \bullet F}\, omv_f}{((((T/T)/\overline{T}_t) \bullet (\overline{\mathbb{R}/\mathbb{R}}_t)) \bullet F}\, \beta_2; \overline{A_1}_t = \overline{A_2}_t = \overline{\mathbb{R}}_t}{((\mathbb{R}/\mathbb{R})/(\mathbb{R}/\mathbb{R})) \bullet F}\, \beta_1; \overline{T}_t = \overline{\mathbb{R}/\mathbb{R}}_t}{\mathbb{R}/\mathbb{R}}\, \beta_1; F = \mathbb{R}/\mathbb{R}$$

This time we successfully concluded that the variable f stands for a unary real function.

8.7.8 OpenMath Signatures

As we have seen, our proposal to add a formal categorial semantics for OpenMath objects to the OpenMath Standard in the future has some consequences for the structure of OpenMath signature files.

- Signatures require standardized vocabulary for categorial *abstraction* and *application* type constructors.
- Signatures of type descriptor symbols require standardized vocabulary for the *type description* notion.

These requirements might be met, for example, by requiring a *type theory descriptor* in a signature file's header. This descriptor would list the OpenMath symbols that denote the three special notions listed above in an OpenMath content dictionary that contains the necessary symbols for describing a type theory for OpenMath.

8.7.9 Compositionality and the Explicit Typing of OpenMath Objects

More radically, the complexities involved in using OpenMath attribution objects to represent explicit typing of OpenMath objects would seem to suggest reconsidering this

approach completely. Recall how earlier on we deduced from the compositionality principle that the special treatment required for a *semantic* interpretation of variable binding induces a requirement for a special *syntactic* constructor for this purpose. The same argument can be made here for explicit typing, which clearly required special handling in our treatment above, with the consequence that special syntax should be introduced into OpenMath to handle it properly – just as explicit typing is done via special syntactic constructors in most existing programming languages.

8.8 Summary and Outlook

In this chapter we have introduced, for the first time, the formal linguistics concept of a *categorial type logic* as surveyed in [24] into the field of content markup language design.

The concept of a categorial type logic, especially in the form of a Lambek calculus that we adapted here to our use, applies to a language that obeys the compositionality principle. In chapter 7 [3] we argued that OpenMath [10] is the first knowledge communication language to meet this requirement; it is thus OpenMath that we apply this new concept to in this chapter, with some success.

- There are clearly problems with the OpenMath Standard 1.0 [10] that we discovered in the course of this exercise and which are now, on our suggestion, in the process of being fixed.
- With these new fixes in place, we are able to propose a basic categorial type logic for OpenMath that matches well with the intended semantics of core OpenMath.
- The basic categorial type logic is a simpler version (without associativity) of the well-known and extensively studied restricted second-order polymorphic **L2** categorial type logic [24], and inherits decidability and other useful mathematical properties from it.
- The interaction between such a categorial type logic and a very general class of "plug-in" lexical type theories has also been studied in the linguistics literature, from which we get decidability of the combined (or rather, layered) type logic given decidability of the plug-in type theory [151].

- Categorial type logics thus allow us to fully specify a categorial semantics for OpenMath, while fully retaining its extensibility both with respect to its "lexicon" (i.e. OpenMath content dictionarys) and with respect to the support for multiple competing type theories. Other type system proposals for OpenMath have been incomplete in that the type semantics of some core OpenMath constructions were not fully specified.

- Our basic categorial type logic proposal for OpenMath nicely clarifies the semantics of OpenMath's important new construct, the OpenMath binding object, especially one with a compound binder as head component.

A price that we have had to pay for this success in the case of OpenMath is that the resulting categorial type logic is non-associative, which would correspond to a loss of currying in an underlying Lambda calculus. However, as we have seen, this price is a direct consequence of specific design decisions for OpenMath that we have thus been able to show would need to be seriously reconsidered in future versions of OpenMath.

A study of consequences of the categorial type logic approach to the design of symbols in OpenMath content dictionary and their signatures has shown that providing signatures for most symbols, including fixed-arity constants, functions, predicates, quantifiers, and operators, but also OpenMath attribute name symbols, is quite straightforward as long as they remain faithful to the core of OpenMath.

However, this study also showed that there are limitations to this approach when considering extensions to that core.

- Variable arity (*a.k.a.* n-ary) function or operator symbols do not work well, though a workaround can be found. However, there is a close analogy to a well-known limitation of such symbols in functional languages like LISP that suggests that the underlying reason for our problems handling this case lie deep, and that the problem may in fact be more with variable-arity symbols in formal languages in general than with our approach in particular.

- While we were able to show that explicit type assignments within OpenMath objects can be handled fairly nicely in the categorial type logics framework, the fit is not exactly perfect. This extension to OpenMath, we find, needs some support concepts to be added to the OpenMath core in order to significantly improve its design.

As a design guideline for future versions of OpenMath in particular, and for all content markup and knowledge communication languages in general, we therefore suggest aiming for a design that supports a categorial type logic **L2** with full support for currying and with those restrictions on second-order quantification that make it decidable [24]. As we have seen, this suggestion has profound consequences for the range of admissible syntactic structures of such languages.

In summary, the concept of a categorial grammar, and the compositionality principle it derives from, have been shown to be a powerful design tool (or design principle, resp.) for content markup languages.

CHAPTER 9

CONCLUSIONS

The title for this dissertation has been "Content Markup Language Design Principles." Living up fully to this ambitious title has, however, proven to be beyond the scope of a single dissertation, and a fuller exploration of design principles for content communication languages is one of the open research questions we are left with.

However, the less ambitious goal for this dissertation that we hope to have achieved is to motivate a particularly promising approach to content markup language design, namely the parallel that we have perceived between natural language and content communication language architectures, and to explore a few aspects of content communication languages that this approach provides us with ready-made tools for. In this dissertation, we concentrated on three such linguistically motivated proposals for an architecture of content communication languages for mathematics which have produced unorthodox but valuable results that already have had an impact on the major standards in this field, namely on OpenMath and MathML-Content. Indeed, our personal influence on both these standards has been acknowledged for several years.

In this dissertation, we have tried to balance an account of the practical experience we gained over the years with implementations and applications of content communication languages on the one hand, and, on the other hand, the more theoretical approach based on the parallel that we propose to draw and follow through between the linguistics of natural language and the architecture and design of content markup languages. In particular, we described in Chapter 5 how the lessons we learned while implementing content communication languages on a variety of software platforms provided the initial intuition that these two fields are intimately related. In Chapter 6, on the other hand, we showed that making this intuitive leap and applying mathematical techniques that had originally been developed for the study of human languages to the analysis of concrete content

communication languages has been of immense practical value, flushing out as it did several deep and subtle common errors in such languages, some of which appear to have escaped notice for years.

If the future goal of content markup languages for the sciences is to continually improve the design and implementations of representations of ever more mathematical and scientific concepts from ever more mathematical and scientific fields of research, it is of paramount importance to try and understand the underlying problems of communication of meaning. Clearly, however, this understanding is unlikely to come solely from within mathematics or computer science or, indeed, computer algebra or artificial intelligence. Improved understanding in this area requires, as we have argued in [2], willingness to understand at least the fundamental results of the (formal) linguistics of human language, since human language is to this day the only known solution to the problem of communicating meaning among intelligent agents, a solution moreover that many authors have convincingly argued has been developing under intense evolutionary pressure over a sufficiently large number of generations that it is likely to have evolved to a fairly high level of quality.

Thus, a thorough problem analysis is overdue. Such an analysis requires considering the full communications model, all the way up from the level of morphology through syntax and semantics to pragmatics and beyond. It must take an external and multi-disciplinary perspective, including input from mathematical logic, linguistics (syntax, formal and lexical semantics, pragmatics, and beyond), semiotics, cognitive science, and the philosophy of language.

As we have seen, the analysis of a fundamental principle of one particular branch of one particular member of this list of disciplines produces with a bit of effort an outline of a solution for a problem in knowledge communication for Computer Algebra systems that it took a highly motivated group of highly competent computer algebraists, computer scientists, and mathematicians several years to reason their way to with sufficient confidence to dare break with traditions of their field.

Thus, we submit that exploring the parallels to a linguistic communications model can give the content markup language design community a jump-start on the road to a deeper understanding of representations of formal "meaning" in scientific environments. In other words: let us "steal," analyze, implement, and test the design principles that linguists over

the decades have discovered in the study of the only high-quality system for exchanging "meaning" known to exist, namely human language, and let us extract those principles that naturally map to communication of meaning between computer systems to more quickly build high-quality systems.

In [2], we argued that such an approach is especially crucial for knowledge exchange between intelligent agents that operate in areas beyond pure mathematics. Thus, for example, we have explored the possibility of using OpenMath as a communication basis for a Problem Solving Environment (PSE) for coupled climate modeling, a multi-disciplinary area of study that involves several fields each of mathematics, computer science, physics, chemistry, biology, geography, meteorology, oceanography, and an open-ended list of future additions. Embedding a useful user interface into such a complex environment alone requires that we study the linguistic and cognitive expectations of users with a wide range of technical backgrounds, and ways of mapping between these and the symbolic and algebraic models implicit in the knowledge communication language employed between the open-ended group of components of the PSE.

Vannevar Bush noted as far back as 1945 [45]:

> Progress is inhibited by the exceedingly crude way in which mathematicians express their relationships. They employ a symbolism which grew like Topsy and has little consistency; a strange fact in that most logical field. A new symbolism, probably positional, must apparently precede the reduction of mathematical transformations to machine processes.

If there's one thing we can learn from the combined history of HTML-Math, MathML, and OpenMath, it is that devising this new symbolism is much harder than it first appears. However, as we have argued, short-cuts do exist.

In summary, we hope to have shown that the study of links to the field of linguistics has already had significant influence on improving the representation of mathematical notions in OpenMath. Indeed, as far as we can tell, OpenMath is the only language of its kind so far that adheres to the Compositionality Principle, a fundamental measure of quality of a language in its field.

We also hope to have made a convincing argument that linguistics and cognitive sciences offer a gold-mine of guidelines for future research into good and useful ways of extending and

improving OpenMath, MathML, and other content markup and knowledge communication languages, guidelines which have the potential to quickly lead to a level of very high quality for extensions to existing content markup languages, in particular those that add support for branches of science that go beyond pure mathematics.

REFERENCES

[1] J. Abbott, A. van Leeuwen, and A. Strotmann. OpenMath: Communicating mathematical information between co-operating agents in a knowledge network. *J. of Intelligent Systems, 1998 special issue: Improving the Design of Intelligent Systems: Outstanding problems and some methods for their solution*, 8(3/4), 1998.

[2] L.J. Kohout and A. Strotmann. Understanding and improving content markup for the Web: from the perspectives of formal linguistics, algebraic logic, and cognitive science. In J.H. Albus and A. Meystel, editors, *Proc. of IEEE Internat. Symp. on Intelligent Control, IEEE Internat. Symp. on Computational Intelligence in Robotics and Automation & Intelligent Systems and Semiotics (A joint conf. on the Science and Technology of Intelligent Systems)*, pages 834–839, Piscataway, NJ, September 14-17 1998. IEEE & NIST, IEEE.

[3] A. Strotmann and L.J. Kohout. OpenMath: compositionality achieved at last. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 34(2):66–72, June 2000.

[4] L.J. Kohout, E. Kim, A. Strotmann, and E. McDuffie. Knowledge networking for decision making about affordability of engineering design with OpenMath protocol support. In *Proc. of IFSA and NAFIPS conferences.*, pages 622–627. IFSA & NAFIPS, IEEE, July 2001.

[5] L.J. Kohout, A. Strotmann, and R. van Engelen. Knowledge engineering methods for climate models. In *2001 IEEE Internat. Conference on Systems, Man & Cybernetics: SMC 2001 Conf. Proc.*, pages 2193–2198. Systems, Man and Cybernetics Society of the IEEE, IEEE, October 2001. CD-ROM Proceedings, IEEE catalog number 01CH37236C.

[6] A. van Leeuwen, M. Roelofs, and A. Strotmann. Objectives of OpenMath – draft version 0.6.4. *OpenMath Reports*, 1995.

[7] J. Abbott, A. van Leeuwen, and A. Strotmann. Objectives of OpenMath: Towards a standard for exchanging mathematical information. In *ISSAC'95 (Poster presentation)*, Montreal, July 1995.

[8] J. Abbott, A. Díaz, and R.S. Sutor. A report on OpenMath. A protocol for the exchange of mathematical information. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 30(1):21–24, March 1996.

[9] J. Abbott, A. van Leeuwen, and A. Strotmann. Objectives of OpenMath. *RIACA Technical Report*, 11, June 1996.

[10] O. Caprotti, D. Carlisle, and A. Cohen (eds.). The OpenMath standard – version 1.0. Technical report, Esprit Project OpenMath, February 2000. URL: `www.nag.co.uk/projects/OpenMath/omstd/`.

[11] P. Ion and R. Miner (eds.). Mathematical Markup Language (MathML). W3C Working Draft. URL: `www.w3.org/TR/WD-math-970515/`, 15 May 1997.

[12] P. Ion and R. Miner (eds.). Mathematical Markup Language (MathML). W3C Working Draft. URL: `www.w3.org/TR/WD-math-970710/`, Jul 10, 1997.

[13] P. Ion and R. Miner (eds.). Mathematical Markup Language (MathML) 1.0 Specification. W3C Recommendation. URL: `www.w3.org/TR/REC-MathML`, 7 April 1998.

[14] P. Ion and R. Miner. Mathematical Markup Language (MathML) 1.0 Specification. W3C Recommendation. URL: `www.w3.org/TR/REC-MathML`, 7 April 1998.

[15] D. Carlisle, P. Ion, R. Miner, and N. Poppelier (eds.). Mathematical Markup Language (MathML) Version 2.0 Candidate Recommendation. URL: `www.w3.org/TR/2000/CR-MathML2-20001113`, 13 November 2000.

[16] D. Carlisle, P. Ion, R. Miner, and N. Poppelier (eds.). Mathematical Markup Language (MathML) 2.0: W3C Recommendation. URL: `www.w3.org/TR/2001/REC-MathML2-20010221/`, 21 February 2001.

[17] *Proceedings of the Second OpenMath Workshop*, Oxford University, July 1994. Institut für Wissenschaftliches Rechnen, ETH Zürich. URL: `www.openmath.org/History/workshops/proceedings-2.html`.

[18] C.M. Sperberg-McQueen and L. Burnard, editors. *Guidelines for Electronic Text Encoding and Interchange (TEI P3), Volumes 1 and 2.* The Association for Computers and the Humanities, the Association for Computational Linguistics, and the Association for Literary and Linguistic Computing, Chicago and Oxford, 1994.

[19] S. Vorkoetter. Openmath specification. Unpublished draft, URL: `www.uni-koeln.de/themen/Computeralgebra/OpenMath/OpenMath.ps`, 1994.

[20] A.C. Hearn and E. Schrüfer. A computer algebra system based on order-sorted algebra. *Journal of Symbolic Computation*, 19(1/2/3):65–79, January, February, March 1995. Design and implementation of symbolic computation systems (Gmunden, 1993).

[21] L.J. Kohout and A. Strotmann. Towards understanding content markup language design. In *11th OpenMath Workshop*, Tallahassee, Florida, November 1998. The Florida State University.

[22] S. Pinker. *The Language Instinct.* William Morrow and Company, Inc., New York, NY, USA, 1994.

[23] T.M.V. Janssen. Compositionality. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 417–473. Elsevier, Amsterdam, 1996.

[24] M. Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam, 1996.

[25] J. van Benthem and A. ter Meulen. *Handbook of Logic and Language*. Elsevier Science Publishers, Amsterdam, 1996.

[26] A. Strotmann. Overview of the OpenMath language structure. In *Proceedings of OpenMath Workshop 3*, Amsterdam, February 1995. RIACA. URL: `www.openmath.org/History/workshops/slides-3/langstruct-all.html`.

[27] *Proceedings of OpenMath Workshop 7*, Trinity College, Dublin, Ireland, 1996. URL: `www.openmath.org/activities/oldws/proceedings-7.html`.

[28] A. Strotmann. Content Dictionaries for climatology. 13th OpenMath Workshop, University of Saint Andrews, Scotland, August 2000.

[29] A. Strotmann. A categorial type system for OpenMath. Computer Science Department, The Florida State University, 2002.

[30] A. Strotmann. Categorial types for OpenMath. In *13th OpenMath Workshop*, Nice, France, March 2002.

[31] J.H. Davenport. A small OpenMath type system. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 34(2):16–21, June 2000.

[32] O. Caprotti and A. Cohen. A type system for OpenMath. Technical report, Esprit Project OpenMath, 1998.

[33] International Organization for Standardization. *ISO 8879:1986: Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*. International Organization for Standardization, Geneva, Switzerland, August 1986.

[34] D. Raggett, A. L. Hors, and I. Jacobs (Eds). HTML 4.01 Specification : W3C Recommendation. URL: `www.w3.org/TR/REC-html40/`, December 1999.

[35] World Wide Web Consortium. Cascading style sheets. URL: `www.w3.org/Style/CSS`.

[36] World Wide Web Consortium. XML Schema. URL: `www.w3.org/XML/Schema`.

[37] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 - W3C Recommendation 10-February-1998. Technical Report REC-xml-19980210, World Wide Web Consortium, February 1998.

[38] World Wide Web Consortium. XML Stylesheet Language (XSL). URL: `www.w3.org/Style/XSL`.

[39] World Wide Web Consortium. XML Linking Language (Xlink). URL: `www.w3.org/XML/linking`.

[40] World Wide Web Consortium. XML Pointer Language (Xpointer). URL: `www.w3.org/XML/Linking`.

[41] World Wide Web Consortium. XML Namespaces. URL: `www.w3.org/XML/#Namespaces`.

[42] HTML background: Members of W3C's HTML Editorial Review Board reflect on the progress of creating HTML 3.2. *World Wide Web Journal*, Winter 1997.

[43] D.E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[44] T.W. Deacon. *The Symbolic Species*. Norton, 1997.

[45] V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, July 1945.

[46] DMV und GAMM Fachgruppe Computeralgebra der GI, editor. *Computeralgebra in Deutschland: Bestandsaufnahme, Möglichkeiten, Perspektiven*. Gesellschaft für Informatik (GI), Passau, Heidelberg, 1993.

[47] E. Kaltofen. Challenges of symbolic computation: My favorite open problems. *Journal of Symbolic Computation*, 29(6):891–919, June 2000.

[48] D. Arnon, R. Beach, K. McIsaac, and C. Waldspurger. CaminoReal: an interactive mathematical notebook. In J.C. van Vliet, editor, *Document Manipulation and Typography. Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography, Nice (France), April 20–22, 1988*, pages 1–18, New York, 1988. Cambridge University Press.

[49] The Unicode Consortium. *The Unicode Standard: Worldwide Character Encoding. Version 1.0. Volumes 1 and 2.* Addison-Wesley, Reading, MA, USA, 1991.

[50] B. Wage. Elsevier's use of HTML, SGML, and math conventions. In *Proc. OpenMath Workshop 4*, University of Copenhagen, Denmark, September 1995. URL: `www.openmath.org/History/Workshops/slides-4/elsevier-all.html`.

[51] N. Kajler. Building a computer algebra environment by composition of collaborative tools. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems, International Symposium DISCO '90, Capri, Italy, April 10–12, 1990, Proceedings*, volume 429 of *Lecture Notes in Computer Science*, Berlin-Heidelberg-New York, 1990. Springer-Verlag.

[52] N. Kajler. CAS/PI: a portable and extensible interface for computer algebra systems. In P.S. Wang, editor, *Proceedings of ISSAC '92. International Symposium on Symbolic and Algebraic Computation*, pages 376–386, New York, NY 10036, USA, 1992. ACM Press.

[53] Wolfram Research, Inc. *Mathlink Reference Guide, Version 2.2*. Wolfram Research, Inc., 100 Trade Center Drive, Champaign, IL 61820-7237, USA, second edition, 1993.

[54] Wolfram Research, Inc. *Mathematica– a system for doing mathematics by computer for Microsoft Windows*, version 2.2.3. edition, 1994.

[55] J. Abbott, S. Dalmas, M. Dewar, A. Diaz, S. Gray, S. Sheridan, A. Strotmann, and S. Vorkoetter. OpenMath communications committee report. *OpenMath Reports*, February 1996.

[56] S. Gray, N. Kajler, and P. Wang. MP: a protocol for efficient exchange of mathematical expressions. In ACM, editor, *ISSAC '94: Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation: July 20–22, 1994, Oxford, England, United Kingdom*, pages 330–335, New York, NY 10036, USA, 1994. ACM Press.

[57] S. Gray, N. Kajler, and P.S. Wang. Pluggability issues in the Multi Protocol. *Lecture Notes in Computer Science*, 1128, 1996.

[58] S. Gray. *A Guide to Programming with MP Version 1.1.3*. University of Kent, 1997. URL: `ftp.mcs.kent.edu/dist/MP/guide.ps.gz`.

[59] O. Bachmann. MPCR: An efficient and flexible chains of recurrences server. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 31(1), 1997.

[60] O. Bachmann, H. Schönemann, and A. Sorgatz. Connecting MuPAD Singular with MP. *MapleTech Journal*, 1998.

[61] S. Dalmas, M. Gaëtano, and A. Sausse. ASAP: A protocol for symbolic computation systems. Technical Report 162, Institut National de Recherche en Informatique et en Automatique, Sophia Antipolis, 1994.

[62] J.A. Abbott. Posso/XDR specifications. Technical report, POSSO Esprit Project, 1994.

[63] A. Strotmann. Computer algebra information interchange and parallel processing in REDUCE, progress report. In *DISCO'93 (Design and Implementation of Symbolic Computation Systems (Demo presented))*, November 1993. URL: `www.uni-koeln.de/\%7Ea0047/disco93/disco93.html`.

[64] A. Strotmann. CALIF – computer algebra information interchange format. In *Proc. of the Second OpenMath Workshop*, Institut für Wissenschaftliches Rechnen, ETH Zürich, July July 1994. Oxford University. URL: `www.openmath.org/History/workshops/slides-2/CALIF-all.ps.gz`.

[65] A.C. Hearn. REDUCE User's Manual, Version 3.5. Technical Report CP 78, Rand Corporation, Santa Monica, CA, USA, October 1993.

[66] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM 3 user's guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, September 1994.

[67] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S.M. Watt. *MAPLE V: Language Reference Manual*. Springer-Verlag, New York, 1991.

[68] H. Hong, editor. *First International Symposium on Parallel Symbolic Computation, PASCO '94, Hagenberg/Linz, Austria, September 26–28, 1994*, volume 5 of *Lecture notes series in computing*, Singapore; Philadelphia, PA, USA; River Edge, NJ, USA, 1994. World Scientific Publishing Co.

[69] P. Marti and M. Rueher. A cooperative scheme for solving constraints over the reals. In H. Hong, editor, *First International Symposium on Parallel Symbolic Computation, PASCO '94, Hagenberg/Linz, Austria, September 26–28, 1994*, volume 5 of *Lecture notes series in computing*, pages 284–293, Singapore; Philadelphia, PA, USA; River Edge, NJ, USA, 1994. World Scientific Publishing Co.

[70] J. Harrison and L. Théry. Reasoning about the reals: The marriage of HOL and Maple. *Lecture Notes in Computer Science*, 698, 1993.

[71] J. Harrison and L. Théry. Extending the HOL theorem prover with a computer algebra system to reason about the reals. *Lecture Notes in Computer Science*, 780:174–184, 1994.

[72] HOL: Higher order logic. URL: `www.cl.cam.ac.uk/Research/HVG/HOL/`.

[73] P. Marti and M. Rueher. A distributed cooperating constraints solving system. *International Journal of Artificial Intelligence Tools*, 4(1–2):93–113, 1995.

[74] P. Marti. *SDRC: Un système de coopération entre solveurs pour la résolution de contraintes non-linéaires sur les réels*. PhD thesis, École Doctorale Sciences Pour l'Ingénieur, I3S, Université de Nice Sophia-Antipolis, 1996.

[75] R.S. Sutor. The OpenMath Consortium. In *NSF Workshop on Remote Collaboration in Mathematics*, Department of Mathematics, Stockholm University, 1995. URL: `www.matematik.su.se/~leifj/wshop/openmath.html`.

[76] O. Bachmann, H. Schönemann, and S. Gray. MPP: A framework for distributed polynomial computations. In Lakshman Y. N., editor, *ISSAC '96: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, July 24–26, 1996, Zurich, Switzerland*, pages 103–112, New York, NY 10036, USA, 1996. ACM Press.

[77] R. Zippel et al. Simlab Mathbus. Computer Science, Cornell University, `www2.cs.cornell.edu/Simlab/papers/mathbus/mathTerm.htm`.

[78] J. Harrison and L. Théry. A sceptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.

[79] C. Ballarin, K. Homann, and J. Calmet. Theorems and algorithms: An interface between Isabelle and Maple. In A. H. M. Levelt, editor, *International Symposium on Symbolic and Algebraic Computation*, pages 150–157. ACM Press, 1995.

[80] K. Homann. *Symbolisches Lösen mathematischer Probleme durch Kooperation algorithmischer und logischer Systeme*. Infix, St. Augustin, 1997.

[81] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning theories - towards an architecture for open mechanized reasoning systems. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 157–174. Kluwer Academic Publishers, March 1996.

[82] M. Fisher and M. Wooldridge. Distributed problem-solving as concurrent theorem proving. *Lecture Notes in Computer Science*, 1237, 1997.

[83] J. Calmet, K. Homann, and I.A. Tjandra. Hybrid representation for specification and communication of mathematical knowledge. In *Proc. ECAI'96 Workshop on Representation of Mathematical Knowledge*, 1996. URL: `faui80.informatik.uni-erlangen.de/IMMD8/staff/Zinn/Ecai/calmet-etal.ps.%gz`.

[84] T.V. Raman. Documents are not just for printing. *Mathematical and computer modelling*, 25(4), 1997.

[85] R.M. Timoney. Style sheet languages and mathematical material. Technical Report TCDM 98-03, Dept. of Mathematics, Trinity College, Dublin, Ireland, 1998. URL: `ftp.maths.tcd.ie/pub/tcdmath/tcdm9803.ps.Z`.

[86] X. Li. Communication of mathematical objects. Master's thesis, University of Western Ontario, 1999. URL: `scl.csd.uwo.ca/~xli/thesis.ps`.

[87] D. Carlisle. OpenMath, MathML, and XSL. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 34(2):6–11, June 2000.

[88] World Wide Web Consortium. Simple object access protocol (SOAP) 1.1. URL: `www.w3.org/TR/SOAP`.

[89] R. van Engelen, K. Gallivan, G. Gupta, and G. Cybenko. XML-RPC agents for distributed scientific computing. In *Proc. IMACS 2000*, 2000.

[90] M. Kohlhase. OMDoc: An infrastructure for OpenMath content dictionary information. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 34(2):43–48, June 2000.

[91] P. Blackburn, J. Bos, M. Kohlhase, et al. Automated reasoning for computational semantics. In *NLDED'99 (submitted)*, 1999. URL: `www.ags.uni-sb.de/~kohlhase/submit/nlded-99.ps.gz`.

[92] M. Maekawa, M. Noro, K. Ohara, Y. Okutani, N. Takayama, and Y. Tamura. OpenXM: an open system to integrate mathematical softwares. In *ISSAC 2000 (submitted)*, 2000. URL: `www.math.kobe-u.ac.jp/OpenXM/issac2000.ps`.

[93] M.R. Genesereth and R.E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, CA, USA, June 1992.

[94] Knowledge interchange format. Draft proposed American National Standard ANSI NCITS.T2/98-004, 1998.

[95] T. Finin et al. Specification of the KQML Agent-Communication Language – plus example agent policies and architectures, 1993.

[96] Y. Labrou and T. Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD, USA, February 1997.

[97] DARPA agent markup language (DAML). `www.daml.org`.

[98] M.C. Dewar and M.G. Richardson. Reconciling symbolic and numeric computation in a practical setting. In *Proceedings of DISCO'90*, pages 195–204, Berlin, 1990. Springer-Verlag, Heidelberg.

[99] R.J. Fateman. How to find mathematics on a scanned page. URL: `www.cs.berkeley.edu/~fateman/papers/see-eqns.ps`.

[100] A. Kosmala, S. Lavirotte, L. Pottier, and G. Rigoll. On-line handwritten formula recognition using Hidden Markov Models and Context Dependent Graph Grammars. In *5th International Conference on Document Analysis and Recognition*, Bangalore, India, September 1999.

[101] G. Gonnet. Teaching mathematics through the Internet: Web Pearls' approach. In *11th OpenMath Workshop*, Tallahassee, Florida, November 1998. The Florida State University.

[102] P.S. Wang. IAMC: Internet accessible mathematical computation. In *Proc. 3rd ASCM*, 1998.

[103] P.S. Wang. Design and protocol for Internet accessible mathematical computation. In S. Dooley, editor, *ISSAC 99: July 29–31, 1999, Simon Fraser University, Vancouver, BC, Canada: proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pages 291–298, New York, NY 10036, USA, 1999. ACM Press.

[104] A. Weber, W. Küchlin, B. Eggers, and V. Simonis. A parallel Gröbner solver as a Web component. In *Proc. CASC'98*, 1998.

[105] A. Weber, W. Küchlin, and B. Eggers. Parallel computer algebra software as a Web component. *Concurrency: Practice and Experience*, 10(11–13):1179–1188, September 1998. Special Issue: Java for High-performance Network Computing.

[106] A. Strotmann. Intelligent interactive learning systems in college-level calculus teaching: An experimental prototype and its lessons. In *Consensus Workshop on Symbolic Notations on the Web*, Tallahassee, Florida, 1999. Florida State University.

[107] A.M. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive.* Springer-Verlag, 1999.

[108] E. Melis. The Interactive Textbook project. In *CADE Workshop 2000 (submitted)*, 2000. URL: `www.ags.uni-sb.de/~melis/Pub/MelisCadews00.ps`.

[109] S.M. Hess, C.G. Jung, M. Kohlhase, and V. Sorge. An implementation of distributed mathematical services. In *Proc. Calculemus'98*, 1998.

[110] T. Berners-Lee. Web future. World Wide Web Consortium, `www.w3.org/Talks/1998/0227-WebFuture/overview.htm`, 1998.

[111] World Wide Web Consortium. Semantic Web activity: Resource description framework (RDF). URL: `www.w3.org/RDF`.

[112] S.S. Dooley. Coordinating mathematical content and presentation markup in interactive mathematical documents. In Oliver Gloor, editor, *ISSAC 98: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, August 13–15, 1998, University of Rostock, Germany*, pages 54–61, New York, NY 10036, USA, 1998. ACM Press.

[113] R.S. Sutor and S.S. Dooley. TEX and LATEX on the Web via IBM techexplorer. *TUGboat*, 19(2):157–161, June 1998.

[114] World Wide Web Consortium. Synchronized multimedia integration language. URL: `www.w3.org/AudioVideo`.

[115] Chemical markup language (CML). URL: `www.xml-cml.org`.

[116] Biopolymer markup language (BIOML). URL: `www.bioml.com/BIOML`.

[117] Universal networking language (UNL). URL: `www.unl.ias.unu.edu`.

[118] R.K. Rew and G.P. Davis. The Unidata netCDF: Software for scientific data access. *Sixth Int'l. Conf. on Interactive Inf. and Processing Sys. for Meteorology, Oceanography, and Hydrology*, February 1990.

[119] OpenGIS. URL: `www.opengis.org`.

[120] International Federation of Library Associations and Institutes (IFLA). *UNIMARC: Universal MARC format*, 1977.

[121] Dublin Core meta-data initiative. URL: `www.purl.org/dc/`.

[122] MPRESS: MathNet.preprints – the mathematics preprint search system. URL: `mathnet.preprints.org`.

[123] T. Berners-Lee. A roadmap to the Semantic Web. World Wide Web Consortium, `www.w3.org/DesignIssues/Semantic.html`, September 1998.

[124] World Wide Web Consortium. Platform for privacy preferences project (P3P). URL: `www.w3.org/P3P`.

[125] G.K. Chesterton. The point of a pin. In *The Scandal of Father Brown*. Cassell, 1935.

[126] M. Kohlhase and A. Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation*, 32(4):365–402, September 2001.

[127] S. Linton and A. Solomon. OpenMath, IAMC, and GAP. Technical report, Department of Computer Science, University of St. Andrews, 1999.

[128] *OpenMath Workshop 3*, Research Institute for Applications of Computer Algebra, Amsterdam, The Netherlands, January 1995. URL: `www.openmath.org/History/workshops/proceedings-3.html`.

[129] M.L. Griss, E. Benson, and G.Q. Maguire, Jr. PSL: A portable LISP system. Technical Report UCP–83, Univ. of Utah, CS Dept., Salt Lake City, May 1982. and ACM Symposium on LISP and Functional Programming, Pittsburgh, Aug. 1982.

[130] A.C. Norman. Compact delivery support for REDUCE. *Journal of Symbolic Computation*, 19(1/2/3):133–144, January, February, March 1995. Design and implementation of symbolic computation systems (Gmunden, 1993).

[131] ANSI Common LISP. American National Standards Institute, 1994. ANSI Standard X3.226-1994.

[132] J. Rees and W. Clinger (eds.). Revised^3 report on the algorithmic language Scheme. *SIGPLAN Notices*, 21(12), December 1986.

[133] J. Marti, A.C. Hearn, M.L. Griss, and C. Griss. Standard LISP report. *SIGPLAN Notices*, 14(10):48–68, 1979.

[134] E.M. Greenwalt, J. Slocum, and R.A. Amsler. *U.T. LISP Documentation*. Computation Center, University of Texas at Austin, May 1975.

[135] A. Strotmann. Die Struktur von Portable Standard LISP 3.4. *Rechenzentrumsarbeitsbericht*, RRZK-90-02, June 1990.

[136] Sun Microsystems, Inc. RFC 1014: XDR: External Data Representation standard, June 1987.

[137] Information Processing — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8824.

[138] Information Processing — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8825.

[139] A. Diaz, M. Hitz, E. Kaltofen, A. Lobo, and T. Valente. Process scheduling in DSC and the large sparse linear systems challenge. *Journal of Symbolic Computation*, 19(1/2/3):269–282, January, February, March 1995.

[140] D. Adams. *Life, the Universe, and Everything*. Pan Books, London, 1982.

[141] H. Melenk and W. Neun. RR: Parallel symbolic algorithm support for PSL based Reduce. Technical report, Konrad-Zuse-Zentrum für Informationstechnik, Heilbronner Str. 10, D-10711 Berlin-Wilmersdorf, Germany, 1994.

[142] R. van Engelen, L. Wolters, and G. Cats. CTADEL: A generator of multi-platform high performance codes for PDE-Based scientific applications. In ACM, editor, *FCRC '96: Conference proceedings of the 1996 International Conference on Supercomputing: Philadelphia, Pennsylvania, USA, May 25–28, 1996*, pages 86–93, New York, NY 10036, USA, 1996. ACM Press.

[143] R. van Engelen, L. Wolters, and G. Cats. Tomorrow's weather forecast: Automatic code generation for atmospheric modellig. *IEEE Computational Science and Engineering*, 4(3):22–31, July-September 1997.

[144] R.A. van Engelen. ATMOL: A domain-specific language for atmospheric modeling. *Journal of Computing and Information Technology*, 2002.

[145] B.H. Partee with H.L.W. Hendriks. Montague grammar. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 5–91. Elsevier Science Publishers, Amsterdam, 1996.

[146] Journal of language, logic, and information, 2001. Special Issue "Compositionality".

[147] K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

[148] K. Ajdukiewicz. Syntactic connexion. In S. McCall, editor, *Polish Logic 1920—1939*, pages 207–231. Oxford University Press, 1967. Translated from [147].

[149] M. Kohlhase, S. Kuschert, and M. Müller. Dynamic lambda calculus. *L.J. IGPL*, 2000. Preprint, submitted.

[150] T.R. Gruber and G.R. Olsen. An ontology for engineering mathematics. In Jon Doyle, Erik Sandewall Pietro Torasso, editor, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 258–269, Bonn, FRG, May 1994. Morgan Kaufmann.

[151] J. Dörre and S. Manandhar. On constraint-based Lambek calculi. In P. Blackburn and M. de Rijke, editors, *Specifying Syntactic Structures*. CSLI Publications, 1997.

BIOGRAPHICAL SKETCH

Andreas Strotmann

Academic Studies and Degrees

Winter 1981/82 – Winter 1989/90: studied Mathematics at Universität zu Köln, Germany.

24 January 1990: conferred academic degree of *Diplom-Mathematiker (Dipl.-Math.)* (M.Sc. in Mathematics) by Universität zu Köln, Mathematisches Institut, Cologne, Germany, with a thesis titled *Implementation von Portable Standard LISP auf der Cyber 180* (Implementation of Portable Standard LISP on the CDC Cyber 180) supervised by Th. Meis.

July 1993 – June 1995: Doktorand (Ph.D. student in Computer Science) at Graduiertenkolleg "Scientific Computing" at Universität zu Köln, with a dissertation project titled "Computer Algebra Information Interchange Format."

Spring 1999 – Spring 2003: Ph.D. student at Department of Computer Science, The Florida State University, Tallahassee, Florida.

May 2003: awarded Ph.D. in Computer Science by the Department of Computer Science, The Florida State University, Tallahassee, Florida.

Academic Honors and Awards

Stipendiat (Fellow), Deutsche Forschungsgemeinschaft (German Federal Research Council), Graduiertenkolleg "Scientific Computing", Universität zu Köln, July 1993 – June 1995.

University Fellow, The Florida State University, Computer Science Department, academic years 1999 – 2002.

Professional Employment

February 1990 – present: Wissenschaftlicher Mitarbeiter (Staff Scientist),[1] Zentrum für Angewandte Informatik, Universitätsweiter Service (Regionales Rechenzentrum) – Center for Applied Computer Science, University-wide Services (Regional Computer Center), Universität zu Köln (University of Cologne), Cologne, Germany.

Fall 2002: Teaching Assistant, Department of Computer Science, The Florida State University, Tallahassee, Florida.

Summer 2000: Research Assistant, Oceanography Department and School of Computational Science and Information Technology (FSU Research Foundation Grant), The Florida State University, Tallahassee, Florida.

Summer 1999: Research Assistant, Department of Mathematics, The Florida State University, Tallahassee, Florida.

Spring 1999: Teaching Assistant, Department of Computer Science, The Florida State University, Tallahassee, Forida.

February 1989 – January 1990: DV-Angestellter (Programmer), Regionales Rechenzentrum (Regional Computer Center), Universität zu Köln (University of Cologne), Cologne, Germany.

December 1987 – December 1988: Werkvertrag (Independent Software Consultant), Regionales Rechenzentrum (Regional Computer Center), Universität zu Köln (University of Cologne), Cologne, Germany

November 1983 – October 1987 Studentische Hilfskraft (Graduate Assistant), Regionales Rechenzentrum (Regional Computer Center) and Institut für Theoretische Physik (Institute of Theoretical Physics), Universität zu Köln (University of Cologne), Cologne, Germany

[1] On leave of absence July 1993 – June 1995 and February 1999 – January 2003.

Journal and Conference Publications

L.J. Kohout, A. Strotmann: "Effective Computations with Distributed Knowledge: The Issue of Compositionality and Ontologies." NAFIPS-FLINT 2002 Proceedings, New Orleans, June 2002.

A. Strotmann: "Principles of Content Markup Languages: Towards Computer-Understandable Documents." The Canadian Journal of Information and Library Science, **26** (2/3), 2001 (extended abstract).

L.J. Kohout, A. Strotmann, R.A. van Engelen: "Knowledge Engineering Methods for Climate Models." IEEE Systems, Man, and Cybernetics 2001 Conference Proceedings, Tucson, Arizona, October 2001.

L.J. Kohout, E. Kim, A. Strotmann, E. McDuffie: "Knowledge Networking for Decision Making About Affordability of Engineering Design with OpenMath Protocol Support". Proc. IFSA/NAFIPS Conference 2001.

A. Strotmann, L. Kohout: "OpenMath: Compositionality Achieved at Last." ACM SIGSAM Bulletin 34 (2000) 2, Special Issue "OpenMath."

L.J. Kohout, A. Strotmann: "Understanding and Improving Content Markup for the Web: from the Perspectives of Formal Linguistics, Algebraic Logics, and Cognitive Science." in: ISIC/CIRA/ISAS '98 Joint Conference on the Science and Technology of Intelligent Systems (IS'98), Gaithersburg, MD, 1998.

J. Abbott, A. van Leeuwen, A. Strotmann: "OpenMath: Communicating Mathematical Information between Co-operating Agents in a Knowledge Network," in: Journal of Intelligent Systems, No.3/4, 1998, Special Issue "Improving the Design of Intelligent Systems: Outstanding Problems and Some Methods for their Solution." = "Objectives of OpenMath" (see below)

W. Antweiler, A. Strotmann, V. Winkelmann: "A TEX-REDUCE Interface." ACM SIGSAM Bulletin 23 (Feb. 1989)

Technical Reports and Articles

J.A. Abbott, A. van Leeuwen, A. Strotmann: "Objectives of OpenMath". RIACA Technical Report 12, RIACA 1996. = J.A. Abbott, A. van Leeuwen, A. Strotmann: "OpenMath: Communicating Mathematical Information between Co-operating Agents in a Knowledge Network." Journal of Intelligent Systems, No.3/4, 1998.

A. Strotmann: "XML 1.0 und MathML 1.0". in: RRZK-KOMPASS: Mitteilungen des Regionalen Rechenzentrums der Universität zu Köln (Communications of the Regional Computer Center of the University of Cologne), Nr. 78 (1998).

T. Bönniger, M. Siegert, D. Stauffer, A. Strotmann: "Monte-Carlo-Simulationen zum Ising-Modell auf Parallelrechnern." ("Monte Carlo Simulations of the Ising Model on Parallel Computers.") Rechenzentrumsarbeitsbericht RRZK-95-03 (University of Cologne Computer Center Technical Report), Dec. 1995.

H. Melenk, A. Strotmann: "REDUCE Version 3.4 User's Guide for IBM System /370 Computers Using OS/VS2 MVS XA." Rechenzentrumsarbeitsbericht RRZK-92-02 (University of Cologne Computer Center Technical Report), Jan. 1992.

H. Melenk, A. Strotmann: "REDUCE Version 3.4 Installation Guide for IBM System /370 Computers Using OS/VS2 MVS XA." Rechenzentrumsarbeitsbericht RRZK-92-01 (University of Cologne Computer Center Technical Report), Jan. 1992.

H. van Uffelen et al., ed.: "Abschlussbericht der Pilotgruppe 'Dezentrale Katalogisierung'." (Final Report of the Pilot Project 'De-Centralized Cataloguing') Universität zu Köln, Dec. 1990.

A. Strotmann: "Die Struktur von Portable Standard LISP 3.4." Rechenzentrumsarbeitsbericht RRZK-90-02 (University of Cologne Computer Center Technical Report), June 1990. URL: `www.uni-koeln.de/\%7Ea0047/struktur/struktur.html`

A. Strotmann, R. Biermann, V. Winkelmann: "REDUCE Version 3.3 Installation Guide for CDC Cyber 180 Computers under NOS/VE." Rechenzentrumsarbeitsbericht RRZK-89-02 (University of Cologne Computer Center Technical Report), March 1989.

W. Antweiler, A. Strotmann, V. Winkelmann: "Typesetting REDUCE Output with TeX – A REDUCE-TeX Interface" (Documentation of the TRI package for the REDUCE computer algebra system, a part of the REDUCE distribution since 1989). URL: `www.uni-koeln.de/REDUCE/tri.ps`

W. Antweiler, A. Strotmann, V. Winkelmann: "A TeX-REDUCE Interface" Rechenzentrumsarbeitsbericht RRZK-89-01 (University of Cologne Computer Center Technical Report), Feb. 1989. (Extended version of the SIGSAM paper listed above).

R. Esser, T. Pfenning, A. Strotmann, V. Winkelmann: "REDUCE Version 3.2 Installation Guide for Control Data Cyber 170 Computers under NOS and NOS/BE." Rechenzentrumsarbeitsbericht RRZK-86-09 (University of Cologne Computer Center Technical Report), October 1986.

R. Esser, A. Strotmann, V. Winkelmann: "REDUCE Version 3.1 Installation Guide for Control Data Cyber 170 Computers under NOS and NOS/BE." Rechenzentrumsarbeitsbericht RRZK-85-02 (University of Cologne Computer Center Technical Report), April 1985.

Conference and Seminar Presentations

A. Strotmann: "Content Markup." Colloquium presentation, Graduate School of Library and Information Studies, McGill University, Montreal, Canada, March 2002.

A. Strotmann: "Categorial Types for OpenMath." OpenMath Thematic Network Workshop, Nice, France, March 2002.

A. Strotmann: "Some Thoughts on Sorted Generalized Quantifiers in OpenMath." Presentation, European Community Thematic Network OpenMath Kickoff Meeting, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, Germany, August 2001.

A. Strotmann: "Compositionality: A Content Markup Language Design Principle." Seminar/colloquium presentation, Graduiertenkolleg Scientific Computing, Universität zu Köln, Cologne, Germany, June 2001.

A. Strotmann: "Towards a Problem Solving Environment for Climate Studies: Perspectives and Some First Steps." Presentation, 2nd Anniversary Workshop of the Climate Institute, a Center of Excellence supported by the Florida State University Foundation, Tallahassee, FL, May 2001.

A. Strotmann: "Intelligent Mathematics Tutoring and Testing: Two Experiments and Their Lessons". Poster presentation, East Coast Computer Algebra Day 2001, Tallahassee, FL, May 2001.

A. Strotmann: "OpenMath Extensions for Climatology." Presentation, 13th OpenMath Workshop, St. Andrews, Scotland, 2000.

A. Strotmann: "Intelligent Interactive Learning Systems in College-Level Calculus Teaching: An Experimental Prototype and its Lessons". Invited Presentation, Consensus Workshop on Symbolic Notations on the Web, Tallahassee, Florida, 1999.

L.J. Kohout, A. Strotmann: "Towards Understanding Content Markup Language Design". Invited presentation, 11th OpenMath Workshop, Tallahassee, Florida, 1998.

A. Strotmann: "The Data-Structure Level and The Expression Level". In: Proceedings of OpenMath Workshop 5, Bath, England, 1996. URLs: `www.openmath.org/activities/oldws/slides-5/data-structures.html` and `www.openmath.org/activities/oldws/slides-5/expressions.html`

J. Abbott, A. van Leeuwen, A. Strotmann: "Objectives of OpenMath: Towards a Standard for Exchanging Mathematical Information". Poster presentation, International Symposium on Symbolic and Algebraic Computation (ISSAC'95), Montreal, Canada, July 1995.

A. Strotmann: "OpenMath Objectives." Seminar presentation, Symbolic Computation Group, Dept. of Computer Science, University of Waterloo, Canada, July 1995.

A. Strotmann: "Overview of the OpenMath Language Structure". In: "Proceedings of OpenMath Workshop 3," CAN / RIACA, Amsterdam, The Netherlands, Jan. 1995. URL: `www.openmath.org/activities/oldws/slides-3/langstruct-all.html`

A. Strotmann, S. Vorkoetter: "Experiences with OpenMath (including demo REDUCE ↔ Maple)". In:" Proceedings of OpenMath Workshop 3," RIACA, Amsterdam, The Netherlands, Jan. 1995. URL: `www.openmath.org/activities/oldws/slides-3/reducemaple-all.html`

A. Strotmann: "Offene Computeralgebrasysteme" (Open Computer Algebra Systems). Zentrum für Paralleles Rechnen and Graduiertenkolleg "Scientific Computing", Joint Seminar, Universität zu Köln, Cologne, Germany, Dec. 1994

A. Strotmann: "CALIF – Computer Algebra Information Interchange Format". In: "Proceedings of the Second OpenMath Workshop, Oxford University, July 1994." Institut für Wissenschaftliches Rechnen, ETH Zürich, Switzerland, 1994.
URL: `www.openmath.org/activities/oldws/slides-2/CALIF-all.ps.gz`

A. Strotmann: "CALIF: Computer Algebra Interchange Format". Projet SAFIR, Institut National de la Recherche en Informatique et en Automatique (INRIA), Sophia Antipolis, France, June 1994, seminar presentation.

A. Strotmann: "CALIF : Computer Algebra Interchange Format." Institut für Wissenschaftliches Rechnen, ETH Zürich, Switzerland, April 1994, seminar presentation.

A. Strotmann: "CALIF : Computer Algebra Interchange Format". PoSSo ESPRIT Project, Dipartimento di Matematica, Università di Pisa, Italy, April 1994, seminar presentation.

A. Strotmann: "Computer Algebra Information Interchange and Parallel Processing in REDUCE, Progress Report." DISCO'93 (Design and Implementation of Symbolic Computation Systems), Gmunden, Austria 1993, software prototype demonstration.
URL: `www.uni-koeln.de/\%7Ea0047/disco93/disco93.html`

Contributions to Other Publications

World Wide Web Consortium: "Mathematical Markup Language (MathML)" (several versions 1998 - present) acknowledges in section 1.2.1 (The History of MathML): "In particular, MathML has been influenced by the OpenMath project [...] The

working group has benefited from the help of many people. We would like to particularly name [...] Andreas Strotmann, and other contributors to the www-math mailing list for their careful proofreading and constructive criticisms." URLs: `www.w3.org/TR/1998/REC-MathML-19980407/` ... `www.w3.org/TR/MathML2/`.

In S. Moss de Oliveira, P.M.C. de Oliveira, and D. Stauffer: "Evolution, Money, War, and Computers - Non-Traditional Applications of Computational Statistical Physics," Teubner, Stuttgart-Leipzig (1999), the book's authors report results they found simulating "an alternative model for child mortality (A. Strotmann, priv. comm., January 1997) based on a reinterpretation of the time scale, [...] taking into account that more growth and thus also more dangers of [DNA copying] errors occur in childhood."

Philippe Marti: "SDRC: Un système de coopération entre solveurs pour la résolution de contraintes non-linéaires sur les réels." Dissertation, Ecole Doctorale Sciences Pour l'Ingénieur, I3S, University of Nice - Sophia Antipolis, France, 1996, describes a prototype application of OpenMath based on several implementations of an OpenMath prototype I provided: "L'auteur de ces modules [OpenMath] est Andreas Strotmann qui est venu au sein du laboratoire I3S afin d'implanter openMath dans le système." (Annexe, Ch. 3, p.79; text in brackets added from context.)