Braille Representation and Mathematics Notation
a comparison of different electronic formats for mathematics and their implications for Braille production

Fredrik Larsson

30th January 2008

Abstract

Contents

Chapter 1

Introduction

1.1 Background

During the last decades, the increase of information available electronically has been enormous. The largest part of documents contain pure text and are not that mathematical in its content. We have a relatively long tradition of representing alphabetic text in a computers and edit it using different word processors or typesetting systems. The well-known ASCII standard for representing characters from the western-European languages has been around for almost 30 years, and before ASCII, other standards for encoding character sets were used. Printers and other text-processing software used these standards to organise the corresponding character glyphs into fonts. It became possible to exchange the documents by sending the files containing the encoded representation of the characters and the receiver could read them if the corresponding fonts were available.

However, when it comes to mathematics, the situation is a bit different. Until recently, there did not exist any standards for representing mathematics notation electronically. Mathematics were often exchanged as images represented in the computer. This approach clearly has some drawbacks:

- It is difficult to edit mathematical formulas or cut and paste them between documents or E-mails
- Documents with mathematical content are not searchable using search engines
- It is not possible to symbolically manipulate the content such as doing computations automatically

Further, not being able to manipulate the mathematical content prevents us from changing the representation of the content, leading to severe restrictions in accessing information for people with reading or printing disabilities and visually impaired or blind people.

To get rid of some of those inconveniences, specialised tools were constructed for manipulating mathematical content or typesetting mathematics. The tools used for doing computations all had their own internal format for representing mathematical expressions. This made interchange between different software systems hard. The same applied for the typesetting tools as well. No real standard that could bridge the gap between these two usages of mathematics was developed. One major reason was that the needs were very different. The tools for computations needed a way to describe the semantics of mathematics, such as programming languages or spreadsheet formats, while the typesetting area mostly were concerned with the presentation and layout of the mathematics. They demanded more advanced fonts and flexibility in the placement of the symbols such as spacing, stretching of operators, raising and lowering the baseline, etc.

Today, the two traditions start to move closer each other, probably due to the heavy use of Internet. It requires a large independence of what tools that are used and it is necessary to be able to search and edit mathematical content, as well as doing calculations with it. This report discusses different ways to represent mathematics in computers and how those formats affect the ability to change presentation for benefit of readers using synthetic speech, braille or large print. The focus is on MathML and LaTeX, two of the most widely used formats for handling mathematical documents. To see how well these technologies satisfies the requirement to change presentation, the focus is on braille output, a media that presents a big challenge due to its properties in conjunction with the notation used in mathematics.

1.2 The Braille System

The Braille system consists of symbols built-up from dots organised in cells. A symbol in Braille often means the Braille representation of a sign, which is a printed character. This terminology can be confusing for readers not customed to Braille but maybe used to use the words symbol, sign and character interchangeably. The representation used in literal braille is 6 dots arranged in a grid, three rows with two dots in each. This gives $2^6 = 64$ different combinations including the space, i.e., the empty character that uses no dots and therefore suitable for representing the inter-word space. This in practise leaves 63 symbols which is a far too low number for representing all different kinds of characters used in written text and is even more insufficient when it comes to mathematics. Further, certain dot combinations are not easy to observe with the fingers when written in certain positions. An example is an isolated single dot surrounded by spaces or close to other single dots. For people losing sight late in the life, and maybe not trained in tactile reading, other combinations may be hard to distinguish as well.

Braille is said to be a linear media since the symbols are organised into lines and lines are placed, equally spaced, on the printed page. The cells also occupy the same space and hence are similar to a mono-spaced font. There are no such things like base-line raising or lowering, boldface or italic typefaces, letters of different sizes, fractions and arrows and so on. This causes trouble for normal text to some extent but it is a far more serious problems when representing mathematics as will be discussed in the next chapter. To handle all this, one needs to construct special symbols not appearing in normal print, such as symbols for baseline shifts, changes in typefaces and font size or even characters denoting geometrical shapes or two-dimensional flows.

To extend the number of symbols that can be represented, multiple cells can be used to denote a single character. However, this can be problematic leading to unambiguities: Is a given symbol a symbol spanning two cells or two consecutive symbols each occupying a single cell? How can I tell where one symbol starts and another ends. One approach is to reserve certain dot combinations as prefixes and make sure that no single-cell symbol may use it as its Braille code. This tends to lead to very long dot sequences for some symbols and hence one has to make sure that such symbols are devoted to infrequent characters. Still, what is infrequent or not is very much dependent on the language and the kind of content being read. We also have the backward compatibility issue that prevents us from doing too comprehensive changes when newer signs shall be standardised and we might want a given sign to have the same representation regardless of the content or language. This is not the case today in most Braille systems due to readability which has priority over simplicity in production.

Another approach would be to have some kind of longest-match approach. This means that as long as Braille cells can be interpreted as belonging to a single sign, it should be considered the same sign. This is known in the computer programming theory as pattern matching in a non-greedy way, i.e., where a pattern match as much as it can instead of as many times as it can. This is a design not very much tested in Braille systems today. The draw-back is of course that if a reader don't know that there exists a symbol spanning three specific cells, he/she might make a mistake treating the pattern as two symbols. It is not clear however if that is especially serious or if it can be considered the same as other interpretation mistakes due to insufficient knowledge in reading and knowing Braille.

In practise, the approach most often adopted is to let a single dot combination mean different things depending on context. The context is given by defining certain indicators used for switching context. The indicators can themselves consist of multiple cells but they always have the semantics of the indicator. Such indicators can be called change-of-semantics indicators. The most well-known example is the digit sign, an indicator that assigns the Braille codes for letters a to j to digits 1 to 0. Termination indicators are those that terminate the effect of an earlier indicator. They can be divided into explicit or implicit. Explicit termination indicators use a termination indicator which is also a reserved symbol. Implicit termination indicators are not indicated by a special symbol. Again, a space terminating the effect of a digit sign is a common example of an implicit indicator. Another example is when a Braille code other than the letters a to j occurs immediately after a digit.

There are also composition indicators used for capital letters or other changes such as typefaces or colours if that needs to be indicated. They can also be terminated by implicit indicators like when a space switches from capital letter mode to normal, or by explicit ones when special symbols are reserved to terminate a change of font, or a base-line shift as with subscripts or superscripts. If modes can be nested or whether termination of a mode implies the beginning of another or return to a previous one are also design decisions that affects Braille the writing system.

Speech is another linear media where things are read continuously in a determined order—an order that sometimes must be very different from the expected flow of perception when reading the printed text.

Printing Braille using Braille printers connected to a computer is the most stream-line production method today. It has been available for many years and works fine for languages using alphabets. The well-established standards for encoding characters are handled by all Braille printers using Braille tables mapping a character to a special combination of dots. This is a bit of simplification though, since the combination to choose depends on context, so usually some kind of translation software/layer is needed. However, very few printers can print mathematics in Braille for various reasons:

- Mathematical software usually generate graphics that is printed by the printer, a method not suitable for Braille printing

- No standard exists for representing mathematics as a linear flow which means that there is nothing for the Braille printer vendors to implement.

- The Braille notation used for mathematics is not as widely standardised internationally as is the symbols for representing plain numbers and letters.

Another goal of this report is to find out if any modifications of the Swedish system for writing mathematics in Braille is necessary. Can the current notation be produced automatically from the leading standards for representation of mathematical texts? Is it possible to evolve the math Braille notation by studying the different ways of representing mathematics in a linear media such as the upcoming standards or file formats used by tools for manipulating and typesetting mathematics?

1.3 Outline

This report is structured as follows: Chapter 2 contains basic theory of mathematics notation. It answers questions like why is the notation system constructed as it is. It discusses how spacing and different styles are used to increase readability and convey complex mathematical content to the reader. The rich way of constructing symbols are shown with many examples, explaining such things as fences, stretchy operators, separators and accents. There is also a thorough theoretic background on operators, introducing concepts like prefix, infix, precedence and associativity. The chapter ends with a comparison of one of the first ways of writing mathematics in a linear system, i.e. the syntax used in programming languages and spreadsheet formulas when typed into the computer.

The next two chapters compare different kinds of electronic formats for non-graphical representation of mathematical expressions. It focuses on two of the most widely used systems, MathML from W3C and TEXdeveloped by Donald E. Knuth. Chapter 3 is a detailed treatment of the presentational mark-up found in MathML. Almost all presentational elements are covered together with examples of how the different conventions learnt from the previous chapter are realised in MathML. Chapter 4 describes another commonly used system for typesetting mathematics called TEX. The underlying model of constructing formulas and expressions are discussed based on the theory in chapter 2.

In chapter 5 there are short overviews of other formats for representing mathematics, both computer software and Braille codes for mathematics are discussed. The report ends with some conclusions drawn from the material in the previous chapters on what consequences and impacts that can be expected on Braille production if MathML or TEX is used in production of mathematical literature. Further, it discusses different approaches for Braille writing systems if ideas from other formats are applied to them. It also summarises some proposals for future work and gives some principles that the author thinks is important for designing a Braille writing system that will let the reader focus on the presented mathematical content rather than the notation used in the Braille copy.

Chapter 2

Preliminaries on Mathematical Notation and Semantics

Mathematics notation is complex by nature. It evolves all the time as people invents new mathematical constructs and it is based on a long experience. It is both a language rich of symbols and a languages that makes heavy use of typographic conventions. As discussed in the previous chapter these two properties makes it extra hard to represent mathematics in Braille or other linear writing system. The following sections will discuss and exemplify used layout conventions as well as the symbolic language in mathematics. It gives a theoretic background on expression construction with operators and operands and discusses how different expression notations give rise to different needs for symbols and layout conventions when writing mathematics.

2.1 Expressions, Operators and Operands

Operands and operators are the smallest part of a mathematical expression. Operands are the numbers, variables, constants and symbols that the operators act upon. Examples include 3, γ, and $\mathfrak{g}$. Ellipsis, like $\ldots$ and $\cdots$ are also counted as operands since they often denote terms, factors or ranges. Operators are the well-known $+$, $-$, $\cdot$ and $\times$ but also others like $\in$, lim, sin, $\pm$ and so on. Operators are classified depending on the number of arguments they require. sin and $\pm$ are unary operators acting on one operand while $+$ and $\in$ are binary and requires two operands. In fact, $+$ can be seen as n-ary operator since one can add an arbitrary number of operands together but one usually classifies it according to the minimum number of operands required. It is not always easy to do a strict classification. In $f(x)$, x is an operand and f an operator but the entire expression $f(x)$ might be an operand in a larger expression.

Operators and operands together form sub-expressions which themselves are considered as operands. Hence, mathematical expressions are built up recursively. Operands can have different types such as numbers, matrices, sets, Boolean values, and vectors. Expressions that makes sense mathematically are called well-defined so not all combinations of operators and operands are permitted even if the number of operands match the operator's required number of arguments. Some typesetting systems for mathematics relies on such assumptions to resolve ambiguities when rendering expressions. TEX is one such typesetting system. Symbols are classified as correct as possible given mathematical knowledge if the author of the mathematical content is not clear enough. Different assumptions made by different systems is what makes exchange of mathematics and conversion between different formats and media so difficult. Just representing symbols don't take into account different definitions used in different parts of the world for the same concept for example.

Most notation languages for dealing with mathematics tries to classify all symbols as either operands or operators. In MathML, the expression $f(x, y, z)$, treats f, x, y and z as operands and the , as an operator that groups arguments together in a certain order. The parenthesis are classified as operands since they are separators or delimiters. In programming languages, classification might be different treating f as an operator, x, y and z as operands and leaving the comma and parenthesis unclassified. In the expression $|x|$, the bars looks like separators but in fact is an operator working on the operand x computing the norm of its argument. To summarise, classification depends on the purpose and context and may be different if one aims for typesetting or if one aims for semantics and wants to verify/evaluate mathematical expressions automatically.

An expression can be modelled as a tree where operators are internal nodes and their operands are leaves. An expression has its sub-expression as child nodes. The number of branches from an internal node depends on the n-arity of the operator. These terms are borrowed from the computer science and some mathematics description languages, e.g., mathML and OpenMath base their functionality on this model. The underlying algorithms in programming languages also relies on such tree models. Other systems, e.g., $\TeX$ base the notion on boxes with symbols or groups of symbols and how they are positioned relative to each other on the page. This model is more suited for typesetting applications. Clearly, the Braille representation that can be achieved from different notation systems varies a lot. Most systems in use today falls somewhere in between the visual model and the semantic model.

2.1.1 Prefix, Postfix and Infix

Most of us are used to see expressions written as $2 + 3 * 5 - 6$ i.e., operands and operators are mixed. This notation is called infix, the operators are written between the operands they act on. Those of you who have used a HP calculator, or programmed PostScript, have come across another notation called postfix. In this notation, the operator is written after the operands, so the expression becomes $3\ 5 * 2 + 6\ -$. The notation is also called "reversed Polish notation". There is also a notation called prefix, with the operator written before the operands like $- + 2 * 3\ 5\ 6$. It is easier for computers to evaluate prefix and postfix expressions than infix but the most noticeable difference is that no parentheses are needed. To summarise, the notation used affects the needed symbols.

2.1.2 Precedence and Associativity

The infix notation can lead to ambiguities if grouping is not used. The most familiar kind of grouping is the use of parentheses, also called fences. We can call such grouping explicit since special symbols are used. Sometimes grouping is implicit such as the two-dimensional arrangement used in a fraction where the numerator is one group and the denominator another. A third kind of implicit grouping is stretchy operators like the radical sign that stretches to indicate what part of an expression that is its operand.

To avoid always using grouping, priorities must be defined between operators. It is not obvious what operator shall be evaluated first if the expression is $2 + 3 * 5$. The 3 is in conflict between $+$ and $*$. To manage this, one defines a precedence between operators telling us the evaluation order. As we all know, multiplication and division goes first and hence have higher precedence than addition and subtraction. Exponentiation has the highest, being evaluated before multiplication and division. That is why $2^2/4$ is 1 and not $\sqrt{2}$. The result of our sample expression is thus 11 and not 19. It is common in mathematical typesetting to have less spacing around multiplication and division than addition or subtraction, to remind the reader about the precedence. Also, the space after the first minus in $-3 - 2$ is less than the space after the second one. This is because the first is a unary prefix operator and the second is an infix binary operator and hence bind less tight showing us that the unary operator shall be evaluated first.

However, precedence is not enough to resolve all ambiguities that may arise in infix expressions. In the expression $2 + 3 - 6$ it doesn't matter if $2 + 3$ is computed first or if $3 - 6$ is computed first. But, how about 2^{3^4}? Is it 2^{81} or 8^4? To answer this, we must introduce associativity which mainly tells us which one of the two operators should go first if two operators with the same precedence collide. In this case, exponentiation is right associative so 3^4 will be computed before raising 2 to the obtained result. Associativity does not lead to any special layout conventions, it just explains why parenthesis are needed in some cases and not in others. It can affect linear rendering, such as braille, if and when parentheses or other grouping constructs are used. This also applies to the subscript operator which is right-associative so that a_{b_c} does not require grouping but the subscript ${a_b}_c$ does. This will be important in a media where script style sizes are indistinguishable from normal sized symbols.

Grouping is what is used to change precedence and associativity. In $\frac{1+x}{1-x}$ both $1 + x$ and $1 - x$ must be evaluated before the division, even while there are no explicit parenthesis and despite the precedence order of the operators. Even if precedence and associativity reduces the need for grouping it cannot eliminate it completely. We also need rules for what operators can follow each other. In mathematics symbols can be juxtapositioned to denote multiplication. It is also the fact that certain operators also works as grouping symbols. Returning again to our norm $|x|$ and an expression like $|a|b - c|d|$ we don't know if it shall be interpreted like $(|a|b) - (c|d)$ or $|a|(b - c)|d|$. Associativity or precedence between norm and implicit multiplication will not help us here. A rule prohibiting implicit multiplication to occur right after the norm will resolve the issue in favour of the former interpretation forcing the latter to require explicit grouping with parenthesis.

2.1.3 Functions and Arguments

The tradition to represent mathematics linearly and without use of different fonts and significant spacing first arised when one started to use computers to perform computations. The programming languages needed constructs to express the mathematical relations and formulas they should compute. Since then, mathematical notation that can be read in a linear order has spread more widely to spreadsheet applications, statistical tools and computer algebra systems as well. Another field of relevance is to find an efficient representation of textual material containing mathematics. Even though electronic typesetting of mathematics is much more efficiently handled if it is expressed as text and not as an image, images or special fonts are used a lot. Very little has been borrowed from the "functional" notation described below. Instead the syntax is very different as illustrated in the later chapters but the semantics and basic ideas are the same.

The adopted solution is to view operators as functions and operands as arguments. Operands can be of different types such as scalars (which are ordinary numbers, constants or variables), vectors which uses one level of indexing, matrices using two level of indexing and tensors that use an arbitrary number of indices. Indexing replaces superscripts since superscripts can be seen as a more efficient way of using symbols or as a function doing table lookup whose name is identical to the vector or matrix and with the indexes as arguments. A vector $\vec{x}$ with components $x1, x_2$ and x_3 could also be written as `x[i]` where `i` is component with index i as in x_i. Operators like addition, division, exponentiation, etc can be handled as functions in a fairly straight-forward way. The expression $1 + \frac{1}{n}$ can be written as `plus(1,frac(1,n))` using nested functions. The notion can quickly be quite hard to read, even short expressions like x^2 requires a function-like expression `power(x,2)`. In practise, most programming languages today do not use a strictly function-oriented approach. Instead, infix notation is used for most operators that can be written on one line such as $+$, fractions are written with /, $\pm$ is replaced by `+-`, and so on. Functions are only used sparingly.

2.2 Layout and Typography

In mathematics, there is a long tradition of experimenting with the possibilities of the printed media to achieve an easier understanding of advanced concepts. Below is a presentation together with examples on how different layout constructs are used to make mathematics more intelligible. Unfortunately, none of these "visual tools" can be utilised in similar ways in Braille since the font and size is fixed as is spacing between symbols and baseline position. The long experience of communicating mathematics has lead to a notation that can be made as complex as necessary for an adequate presentation. Still, the recursive nature of expressions described in the previous section can be recognised in the typography as well.

The complexity ranges from very simple mathematics expressions like $2+3$ to complex notations but still the notation follows the logic structure of mathematics. Simple sub-formulas can be combined into more complex formulas which in turn can be nested and concatenated in a nearly infinite number of ways using the building blocks of mathematics such as operators, functions, grouping symbols and so on. To get a fully functional representation in Braille, the Braille mathematics notation must contain counterparts for these properties as well.

2.2.1 Font Styles, Sizes and Spacing

Mathematics textbooks make heavy use of different fonts and their variations. It is easier to find out what is mathematics and what is surrounding text explaining the mathematics if font styles differ between the two. The following list contains the most common font families and variations used:

- Normal upright text as in This text.
- Slanted text as in *This text.*
- Italic text as in *This text.*
- Boldface text as in **This text**.
- Text in Black-board font as $\mathbb{Q}, \mathbb{R}, \mathbb{Z}$.
- Text without serifs like in This text.
- Different symbols that are letters written with special typefaces as in $\daleth$ or $\mathfrak{a}$.
- Type-writer text as in `This text`.

- Small capitals as in $\textsc{This text}$.

To differentiate the variable named xy from the product of the two variables x and y written as xy one often uses italics text for single-letter constants and an upright font for variable names consisting of multiple letters such as function names like $\sin x$. Further, the italics style used is not exactly the same in a math context as in plain text. Digits and punctuation characters are normally written in upright text as well.

Font sizes may vary as well. This is especially true in subscripts and superscripts of various complexity. Except from a default font size there is a need for a smaller font size for subscripts and superscripts as x_1^n and even an additional smaller one for scripts, used when sub- or superscripts appears in many levels, as 2^{3^4}. The more levels, the better typesetting, consider $2^{2^{2^x}}$. It is also possible to use scripts with the same size regardless of level, x^{y^z}. This is however not considered to be good typesetting of mathematics. Delimiters such as brackets and braces can when nested inside delimiters of the same kind be slightly varied in size such as in $f\bigl(s(f(x)+g(x))\bigr)$. Not counting the bigger versions, there are still over 20 delimiters with such stretched versions that are frequent in math literature. This is also a kind of font size variations.

Font sizes, or sizes of certain symbols, also varies if the mathematical formula is typeset in inline-style, or in display-style. In fact, these two styles should be considered more like two different environments since not only the font size differs. Inline style is when the formulae is embedded in the text with surrounding text occuring on the same line while display-style is used when the formula appears alone on a line. Operators that looks differently in the two environments are called large operators. Examples are $\sum$, $\prod$, $\int$, $\oint$, $\otimes$ and some more which in display-style will look like $\displaystyle\sum$, $\displaystyle\prod$, $\displaystyle\int$, $\displaystyle\oint$, $\otimes$. Fractions may be written with slanted bar instead of a horizontal one in in-line style with a slightly lowered denominator and raised numerator.

The environment also affects things like placement of subscripts and superscripts on operators with movable limits like products, boundary limits, integrals and sums. The convention is usually to write $\int_0^{\frac{\pi}{2}} \sin x\,dx$ in inline style and $\int\limits_0^{\frac{\pi}{2}} \sin x\,dx$ in display style. Some typesetting systems automatically selects the appropriate display of the limits but if the subformulas are complex one might prefer to always use limits to move them away from the integrand or summand. Generally, scripts above or below a symbol is called limits to differentiate them from subscripts and superscripts that appears to the right. They can also appear to the left of the affected symbol and are then called prescripts. The term movable limit comes from the fact that we semantically want to call all expressions in script positions limits in context of integrals and sums rather than subscript or superscript even if the latter more reflects how they are entered on a computer.

Some symbols like certain operators, and accents vary in size depending on their scope, or rather the size of the expression they apply to. Examples include integrals, $\int_0^\infty f(x)+g(x)\,dx$ and radicals, $\sqrt[3]{1+\sqrt{x-\sqrt{u}}+\sqrt{\sqrt{x}+1}}$. The same is true for summation and product symbols, $\sum_{i=0}^n \prod_{j\in R^+} c_j x_{i-j}$, and for set and logical symbols, like $\cup$ and $\bigcup$. These symbols are said to have stretchy properties. More examples are shown in the next section.

Other stretchy operators like parenthesis, brackets and braces can be stretched in different directions, either symmetrically or assymetrically as the parenthesis in matrix multiplication of non-square matrices or brackets in equation systems (see next section.) Another name for such enclosing symbols are fences.

Another example is the "hat" symbol for denoting approximations $\hat{x} \approx x$ that has a wider form as in $\widehat{x+y} \approx \widehat{\hat{x}+\hat{y}}$. This also holds for $\tilde{x}$ and $\widetilde{xyz}$. In general, accents serves a different purpose than in litteral text. In mathematics they are applied to entire expressions to indicate some mathematical property while accents in printed text most often acts on a single symbol to change its pronunciation. Accents or bars around symbols or formulas are called embellishments.

Spacing is used to remind the reader of precedence of binary operators. In the simple formula $(a+b)^2 = a^2 + b^2 + 2ab$, the space around $+$ is larger than that between a and b, in $2ab$ due to the higher precedence of multiplication over addition. For operators that are not in infix form, space occurs only to the left of a prefix operator and to the right of a postfix one. Spacing is also different depending on if the operator is in a script position or not, and if the environment is displayed or inline. How many different widths of spaces that are available depends on the typesetting system.

Related to spacing is line-breaking whose formatting rules are formulated with respect to the form of an operator. It is allowed to break a line immediately before or after a traditional operator in infix form like $+$ but not considered a good formatting convention to break a line immediately after a (,) despite the fact that it is an infix operator. This means that renderers should distinguish operators that act like separators from ordinary operators.

2.2.2 Two-Dimensional Constructs

Printed material, mostly containing text, is linear in its nature. Characters are grouped into lines and lines are laid out after each other forming pages. In principle, there is a one-dimensional reading flow where there clearly exists a linear order between the characters and words. This is similar to Braille as well. Mathematics notation is two-dimensional in its nature. There are fractions, subscripts, superscripts, underscripts and overscripts, matrices, etc.

Fractions can be very simple such as $\frac{1}{3}$ or compound as

$$\frac{\frac{a}{b}}{\frac{c}{d}}$$

where the main bar is thicker that can be the case in elementary textbooks on mathematics. There are also more complex ones like continued fractions:

$$\frac{a_0}{b_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2 + \cdots + \frac{a_n}{b_n}}}}$$

There are also fraction-like constructs where expressions are stacked upon each other with no fraction bar like binomial coefficients $\binom{n}{k}$.

Subscripts and superscripts can be very simple such as in H_2O or 12 m^2 and more complex such as

$$\sum_{\substack{i,j=0\\ i>j}}^{n^2-1} a_{i,k_1^2} b_{j,\delta\prime}$$

In tensor calculus they may be staggered as in $R_i{}^{jk}{}_l$.

Writing matrices and determinants also involves constructs spanning multiple rows but the boundary between each row is distinct as in the following 2×3 matrix:

$$\begin{matrix} 1 & a & 5,9 \\ 2,1 & 4 & bc - d \end{matrix}$$

equation systems and function definitions are also typeset utilising many rows to make them easier to follow. Symbols are also stretched over multiple rows to show grouping as in

$$\left\{ \begin{matrix} 2x + 3y = 9 \\ 5x - 6y = 9 \end{matrix} \right\}$$

and

$$|x| = \left\{ \begin{array}{ll} x, & x \leq 0 \\ -x, & x < 0 \end{array} \right.$$

Other symbols are stretched as well such as in the characteristic polynomial $\chi(\lambda)$ of a 2×2 matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

given by the determinant

$$\begin{vmatrix} \lambda - a & b \\ c & \lambda - d \end{vmatrix}$$

A common name for all of these constructs is array constructs.

Arrows are used to describe directions of flows as in logical implication $a \longrightarrow b$ or more advanced flows like in illustrating elementary division. Text and other symbols may be stacked above each other as in the chemical formula $H_2CO_3 \overset{Heat}{\longrightarrow} H_2O + CO_2$. Sometimes, geometrical shapes in different sizes are used as operators like $x \circ y \bullet z$.

2.2.3 Symbols and Semantics

The most important aspect of mathematical notation is its flexibility. Basic notation for well-known mathematical constructs very seldom changes and still the notation system can keep up with the new research and innovations mathematicians make around the world. Besides from the tight correspondence between notation structure and mathematical structure, the rich ways of constructing new symbols plays a significant role in this case. Except for utilising a large number of letters, new symbols are created by subscripting or superscripting them with other symbols. These script symbols may in turn consist of letters and other scripts or diacritical marks. The letters a through z and A through Z are the most common. Some have variations such as ℓ used instead of l to distinguish it from digit 1.

The most common diacritical mark or accents illustrated together with letter a are $\hat{a}$, $\check{a}$, $\tilde{a}$, $\acute{a}$, $\grave{a}$, $\dot{a}$, $\ddot{a}$, $\breve{a}$, $\bar{a}$, $\vec{a}$, $\mathring{a}$. These accents can be placed over entire formulas as well such as in $x \bar{+} y$ or $\overline{x + y}$. When i and j are used with accents, reading may be improved by using dot-less counterparts $\imath$ and $\jmath$. You can even have accents on top of accents over a single letter like $\hat{\hat{a}}$ or typographic variations such as $\hat{\hat{a}}$ and $\hat{\hat{a}}$. Most of these accents have corresponding symbols in normal text but spacing are usually different.

The examples above showed variations on the Latin letters but they are not the only one used. Greek letters, small and capital, like α, β, γ are used as well and some of them have special variants like ϕ and φ, θ and ϑ, ρ and ϱ, ϵ and ε. Hebrew letters like $\aleph$ and $\gimel$ for transfinite cardinals might occur as well. There is also a large number of operators in use and various other "odd" symbols that are neither letters or digits, nor compound symbols with scripts or accents.

To show patterns in complicated expressions it is common to use ellipses. They also exists in some different variants, $\ldots$, $\cdots$, $\vdots$ and $\ddots$. There are rules for their usage in good mathematical typesetting. If the ellipsis denotes a term such as in $x_1 + x_2 + \cdots + x_n$ one uses the centred dots but if the ellipsis denotes a range as in $x = 2k, k = 1, 3, \ldots$ it is the left justified dots that should be used. These kind of symbols are often used with separators like ordinary punctuation characters.

There are also many symbols that look similar but must be distinguished when doing a correct translation in Braille or mark-up. For example, the Greek letters ν and κ must be distinguished from the italic letters v and x. The letter ϕ should not be confused with the slashed zero ($\emptyset$) to denote an empty set. The lowercase epsilon, ϵ is different from the symbol to denote set membership ($\in$).

Chapter 3

MathML

3.1 Basics

The popularity of the World wide web proves the fact that there is a great need to exchange information between different parts of the world. This was also the main goal when Cern developed the HTTP protocol and HTML format in 1990 though they mostly considered scientific documents and research institutes. The large use of HTML made it evident that the approach was right but the lack of representing mathematics in HTML was a severe short-coming. For various reasons outlined in chapter 1, images are not a suitable format. This was the main motivation for designing MathML, Mathematics mark-up language. It is very similar to HTML, using tags and attributes and can only represent mathematics since the surrounding text and layout is meant to be expressed in HTML. Hence, there has been an important principle that it should integrate well into HTML. Today, the format is of version 2.0 and it is supported in most major browsers and often embeded through the mechanism of XML namespaces and XHTML.

However, there was also other goals for MathML. One was the exchange of mathematical formulas between different document-processing tools or cut-and-paste of equations. This is a form of communication between different software and naturally, other such communications were also considered. One was converting to and from other formats such as OpenMath and TeX. Further, exchange between different algebra packages was another major goal and the reason why MathML has two kinds of elements. One is called presentation elements and deal with the presentation of mathematics. It is sufficiently structure-oriented to make certain semantic reasoning feasible. This set of elements is the subject of this chapter. The other part, called content element is suitable for automatic evaluation and is more of semantic nature than presentational. They will not be treated any further in this report though.

3.2 Content and Presentation, Two in One

The goals of MathML requires the language to be able to represent two aspects of mathematics. The requirement that it should be used to exchange text with mathematical content means that the language should be able to convey the layout of mathematical notation. Further, to satisfy the requirement that MathML could be used to exchange mathematical data between different computer algebra systems means the mark-up should also be able to represent the semantics of mathematical notation. It must be possible to evaluate MathML formulas. This is not possible just by examining the layout and structure of the used notation. Therefore, there are two different sets of tags in MathML. The first set are called presentation elements and start with the letter m. The second group is of the mathematical content, the semantics, and most of them start with a c. This approach is a bit different from HTML where the tags represent structure and content; and the layout is expressed using styling languages like CSS and XSL. In MathML, the presentation tags are sufficiently abstract and general to be media-independent and hence the term content means the meaning of the mathematical expressions. Style languages still serve a purpose together with MathML – namely to show other visual effects, not strictly mathematical, and to style the surrounding text and images in the document. It is possible to mix both kinds of mark-up yielding a document that are useful in many different situations. This chapter will not deal with the content elements, just the presentation tags.

3.3 The MathML Presentation Elements

3.3.1 Operands and Symbols

MathML uses so-called token elements to mark up operands. Operands are numbers, constants, variables and function names. The used elements are `mn`, used for numbers, and `mi` used for the rest. Single-letter constants are rendered in italics and constants and variable names with multiple letters are rendered with a normal upright font like numbers. Note that even if most constants and variables indirectly may be interpreted as a number, constants like π and e should be tagged using `mi`. On the other hand, there are numbers that should not be tagged with `mn` but rather using elements for more complex notation such as rational numbers like $\frac{2}{3}$ or complex numbers like $2 + 3i$. Roman numbers, or Arabic numbers written with letters should use `mn` as well.

When writing symbols in MathML, one can use all characters available in Unicode. They can be written directly in UTF-8 and such character encodings or expressed as entity references like &pi; or &exponentialE;. The `mi` element also takes an attribute called `mathvariant` that take values like bold, bold-italics, fraktur, etc. They are used to indicate that a specific font selection is important for the meaning of the symbol so that document-wide style changes does not destroy the meaning of a given formula. For proper mark-up, always try to use Unicode characters if they exist before using "math variants" of other alpha-numeric characters that are visually the same. This makes it easier for automatic interpretation which is important for braille translation and speech rendering.

If the available symbols is not enough, there is an element that can be used to access symbols from other fonts. This construction shall only be used if nothing else can be used but it is the extension mechanism available for reaching other symbols not yet in Unicode. The element is named `mglyph` and is used inside the token elements where content is allowed. The two attributes `font-family` and `index` specifies the intended glyph. The index is an integer pointing out the addressed glyph relative to the font named by the font-family.

3.3.2 Operators, Separators, Fences and Accents

MathML uses the `mo` element to represent operators. Not only traditional operators like $+$, $-$ or $*$ are treated as operators, but also any other token that is not a number or an identifier. This means that separators like (,) or (;), fencing symbols like | and parenthesis of various kinds and accents are classified as operators as well. On the other hand, the ordinary division operator (fraction bar) is not marked up with mo, it is implicit in mfrac. Elementary functions and radicals also get special treatment as discussed in other sections.

The position of the operator inside a horizontal group (mrow) determines if its form is prefix, postfix or infix. This automatic rule can be overridden using the form attribute. Grouping is discussed in a later section. There are also various other attributes associated with the mo element. Spacing around the operator is controlled by the two attributes lspace and rspace respectively. There are also Boolean attributes for controlling movable limits and stretching properties. The kind of content of mo, i.e. if it is an ordinary operator, fence symbol, separator or accent is available as other Boolean attributes. The values for all these attributes very seldom needs to be set manually. Instead, the operator together with its form are used as a key to an operator dictionary containing appropriate values for the operator's attributes.

MathML also has a set of invisible operators. They are invisible in graphical rendering but contain useful information for other kind of renderers. For example, the entity &invisibletimes; exists and should be placed infix where implicit multiplication is assumed but not explicitly shown. Another similar entity is &applyfunction; that signals that a symbol is a function name and not a variable multiplied with an expression inside parenthesis. These constructs are useful for speech rendering of MathML.

3.3.3 Sub-Expressions and Horizontal Grouping

As discussed in chapter 2 expressions are constructed from their building blocks which are operands and operators but also smaller sub-expressions. One form of grouping is parenthesis that changes evaluation order from what should be default according to precedence and associativity but grouping in MathML occurs for other reasons as well. Rendering can get higher quality since spacing can be affected by grouping and more "intelligent" decisions on line-breaking can be accomplished. Mark-up can also be easier to generate and validate. For example, the element that describe a fraction can be thought of as taking two arguments, one group for numerator and another group for denominator, instead of taking an arbitrary number of arguments, where one of them is an infix division operator. The same applies to other constructs as well such as square roots and exponents.

Grouping is done with the mrow element that can be nested and contain arbitrary child elements from MahtML. To write $\sqrt{5+x}$ you type `<msqrt> <mrow> <mn>5</mn> <mo>+</mo> <mi>x</mi> </mrow> </msqrt>`, spaces are not significant since it is handled by the MathML renderer. It is not allowed to group operators and operands together if they should not be rendered on the same row, that is probably why the element is called as it is, mathematics row. Grouping also affects spacing and the quality of the rendering increases if operands are not grouped with other operators than those who applies to them. It should not be considered as a parenthesis. Rendering quality is not only a visual thing, durations in time when reading math aloud are often used to convey spacing information, or what belongs to a numerator or is the base or index of a subscript etc. It also allows the presentation mark-up to be used to some extent in semantic interpretation such as algebraic applications where content mark-up elements are suggested.

Since MathML should be a base framework for interactive math, grouping can also be used to express a hierarchy of the parts in an expression to make it possible to reference the different parts, or perhaps show different parts in different colours as the discussion in the surrounding context focus on them. A less obvious usage of grouping is to nest mrow elements when coding function arguments as in $f(x, y)$. The preferred code is `<mi>f</mi> <mo>&af;</mo> <mrow> <mo>(</mo> <mrow><mi>x</mi> <mo>,</mo> <mi>y</mi> </mrow> <mo>)</mo> </mrow>`. This allows for a non-graphical rendering of the form “f of x and y” as well as referring only to the arguments not taking notice of the parenthesis.

Another element which acts like some kind of grouping is the `mphantom` element. It makes the sub-expression inside it invisible, still reserving as much space as would have been occupied. This can be used for alignments like

$$\frac{x+y+z}{x\phantom{{}+y}+z}$$

which is different from

$$\frac{x+y+z}{x+z}$$

but its use in non-visual rendering is probably limited. Another element is `menclose` that can be used to draw circles around, or lines through, expressions. The behaviour is controlled by attributes. This element may require special treatment in non-visual rendering such as Braille.

3.3.4 Fractions and Radicals

A fraction in MathML can be described as `<mfrac>numerator denominator</mfrac>` where the numerator and denominator each stands for a sub-expression, each grouped within an `mrow` element of course. The `mfrac` element also takes an attribute `bevelled` which indicates if the fraction bar should be horizontal or slanted. Other attributes control line-thickness, useful for nested fractions, and alignment of denominator and numerator relative to the bar. The element is also used for fraction-like structures such as binomial coefficients, $\binom{a}{b}$ or $\binom{a}{a_1,a_2}$, Legendre symbols like $\left(\frac{a}{p}\right)$ and Eulerian numbers $\left\langle{n \atop k}\right\rangle$. This means that it is not the semantics that is marked-up, rather it is the visual structure. This may have accessibility issues since there will be no standard way to tell if something using `mfrac` is a fraction or a binomial coefficient for example. The MathML content elements may give more precise indication.

Radicals are coded with two different elements, `msqrt` and `mroot`. The first is used for square roots and takes one sub-expression using `mrow`. The latter is used for other roots and hence have the form `<mroot> base index</mroot>`. The base is the expression inside the radical and the index is the root to compute, e.g. 3 for cubic roots.

3.3.5 Scripts and Limits

MathML differs between scripts placed under and over symbols, from scripts placed in the usual subscript/superscript position, attached to the base expression. For layout-oriented mark-up, it could have been enough with just one element and attributes or styling mechanisms to control the placement of the scripts but more semantics can be represented if one separates subscripts from scripts written under a symbol, such as accents, or scripts written as in tensor notation, as vertically aligned pairs. MathML also treats prescripts, scripts appearing immediately before a base expression, specially using its own element. Further, the scripts are not just attached to the last symbol in an expression, like in TEX, instead the entire base expression is included in the mark-up. This give both rendering a higher quality as well as allowing semantic interpretation and some evaluation of mathematical formulas coded with presentation elements as well.

Subscripts and superscript expressions are coded with the `msub` and `msup` elements. The syntax is `<msub> base index </msub>` where index is set in a script style, slightly smaller than the style used for the base expression. base and index are grouped in `mrow` elements as always. The syntax is analogous for `msup`. There is also an `msubsup` element that attach both a subscript and a superscript to the base expression, syntax `<msubsup> base sub sup </msubsup>`. The scripts are rendered like P_2^2, not staggered as $P_{2}{}^{2}$. For scripts under and over a base expression, like limits on large operators like sums and integrals, one uses the similar elements `munder`, `mover` and `munderover`. They have corresponding syntax as the previous three and are also used for accents.

For tensors and prescripts one uses the `mmultiscripts` element. It attaches vertically aligned pairs of indices, and prescripts to its base. The syntax is more complex than for the earlier ones: `<mmultiscripts> base (sub sup)* <mprescripts/> (sub sup)* </mmultiscripts>`. As can be seen, the order is that the base expression is given first, then scripts to the right of the base and finally, after the `mprescripts` element, more scripts. The scripts are specified in pairs, subscript followed by superscript, and the * indicate that there can be arbitrary many of these pairs, including no pair, for example when only prescripts are present. If any index in a pair should be omitted the empty element `none` is used in its place.

3.3.6 Changing Size, Style and Spacing

MathML provides an element `mspace` to insert an arbitrary sized white-space and an element `mpadded` to adjust the size and placement of the bounding box surrounding its content. They should be used to increase reading quality where the default spacing rules will not give the desired result. When specifying space or padding both relative and absolute measures can be given. There are also 7 pre-defined names that can be given, ranging from "veryverythin" to "veryverythick". Each of the 7 sizes can be preceded by the word "positive" or "negative" as well to decrease or increase the existing space. The absolute values the 14 names denote are also changeable in the document.

There are also a lot of warnings of miss-usage of these elements given in the specification. They should NOT be used to optimise spacing for a particular renderer or solve problems that are caused by bugs in MathML browsers. They should NOT be used for alignment or other constructs where more appropriate MathML constructs exists, such as aligning expressions where the `mphantom` element is a better alternative. Such usage destroys the use of MathML as an exchange format and media independence. Another forbidden usage is to combine symbols with space and padding adjustments to create new symbols. The mathematical meaning must remain even if all user-inserted spacing and padding are removed from the document before rendering.

A more used construct is the `mstyle` element, used to change style in the document. In fact, it does more than that. It acts like the `div` and `span` constructs in HTML since it can also be used to change inheritance or default values of other attributes that doesn't have anything to do with style at all, such as operator form (pre-, post- or infix). Since external style sheets are allowed, there are many fonts to choose among and the common typefaces discussed in chapter 2 are present. Intrinsic in MathML one can choose between the environments display-style or inline as well. The number of script sizes is not fixed but depend on the font size which is divided or multiplied with an amount when the scriptsize is increased or decreased respectively. This amount can be set, as well as a minimum limit for the font size in scripts and other constructs that automatically reduces the font size, such as fractions or radicals.

As always, some words of warning is in its place. It is considered bad coding to change font family on a symbol to visually get another symbol. Again, this destroys automatic interpretation by computer algebra systems. If something forces any of the bad usage scenarios given in this section, provide semantic elements from the content tag set so the meaning of the visual effect will be exactly defined.

3.3.7 Tables and Matrices

MathML provides table elements much like the table elements in HTML. The `mtable` element surrounds the table or table-like structure. Each row is wrapped into `mtr` elements, or `mlabeledtr` if rows need to be referenced or numbered like in equations. The placement of labels, to the left or right of the equation, can be expressed as well. Each entry, or cell, is placed inside a `mtd` element. With table-like notations, such things as aligned equations, commutative diagrams, matrices and blocks of matrices, determinants, etc are also considered. To accomplish rendering of such varying structures the elements have many attributes regarding layout. Frames around tables are controlled separately from internal borders between cells. Entries could span multiple rows or columns and width and height of each `mtd` can be controlled individually. There are also special alignment elements to partition cells into groups and control how these groups should be aligned. A visual renderer inserts space in the table to achieve the effect, an aural renderer could use these elements to change the rhythm for example.

3.3.8 Elementary Functions

MathML does not include any of the elementary functions like sin, log, arccos, etc among its presentation elements. They exists among the content elements for encoding semantics but in presentation mark-up they should be written as ordinary text within an `mo` element. The `mtext` element would not be appropriate since it will not communicate the fact that the text is a kind of operator.

3.3.9 Miscellaneous Constructs

Most presentation elements of MathML have been covered in this chapter. The ones that remains are covered shortly here. The `mtext` element is used to include arbitrary text inside a mathematical expression such as the word “where” in a definition like the following:

$$\begin{array}{lll} f_n = 1 & where & n < 2 \\ f_n = f_{n-1} + f_{n-2} & where & n \geq 2 \end{array} .$$

The `ms` element is similar but should be used when storing literal strings intended for programming languages using MathML to carry information. The `merror` element is used to carry erroneous information in a MathML element and still keeping the result as valid MathML. It can contain bad mark-up or mathematical error messages in plain text that occurred when some tool processed MathML input.

Since MathML is an electronic format it also has mechanisms for interactive facilities not useful for printed media. All mathML elements accepts ID attributes so they can be targets for hypertext links or be links themselves. This makes it possible to point to definitions of notations as it is used, or refer back to earlier derived results in the content. There is no built-in cross-referencing scheme for automatic numbering or label expansion, so this must be handled by an external style sheet language such as XSLT.

Another element is `maction` to assign actions to sub-expressions. For example, the reader can select different sub-expressions and the author can assign explanations in text that appears in the MathML browser as tool tips or messages on the status line, explaining the meaning of the expression in detail. Actions can be assigned to groups of sub-expressions as well and it is possible to toggle between which of them should be displayed. This can be used to mark up exercises and their answers or to expand ellipses or other sub-expressions to a more detailed expression which let a reader view complicated equations hierarchically or follow a logical inference or proof step by step.

Chapter 4

The TEX Typesetting System

4.1 Basics

The TEX system was constructed in the beginning of 80s by Donald E. Knuth, a mathematician and computer scientist who should typeset his books and could not find a typesetting system on the market that was able to typeset his manuscripts in a satisfactory way. Thus, he constructed one himself. Despite this background, it is a fully fledged typesetting system that even though the main users are scientists, can typeset any kind of document. However, it is most well-known for its ability to typeset mathematics in a readable way, without requiring the author to express all details of spacing, font size, font style, etc. A more well-known version is LATEX which is a macro package written in TEX that simplifies some constructs such as section numbering, cross-referencing, font selection and some mathematical constructs. LATEX was constructed by Leslie Lamport in the end of 90s.

The language is a macro language and users can write their own packages that customise the system to their on needs, providing document-specific or organisation specific styles and commands. Very few would probably write in "plain TEX " today since it is too low-level, but its commands are there if needed for precise control. The language is similar to a programming language in the sense that the document is described in a text file and a compiler-like tool transforms it to a device-independent representation of pages. This file is then used together with fonts to produce a media-specific version to print, or display on the screen, or something else. TEX also has a companion called `metafont` that can be used to construct fonts. Since the text you type in the text file are references to glyphs in the fonts, you can typeset whatever you want, as long as the font metric is known so that line breaks and page breaks can be computed successfully.

The language is in most cases very simple to read. Basically you write the text you want to typeset and mix it with control sequences to reach other symbols not present on your keyboard. You can also define your own control sequences. Plain TEX comes with a large amount of control sequences and fonts for mathematical symbols from start and if you want to use other extension packages they document their own commands and fonts used. The language is not XML, it is rather like RTF. The \ symbol is used to start a command and {, [,] and } are used to pass arguments to the commands or to construct groups that limit their scope. The rest of this chapter will give an overview of how different mathematical constructs are expressed in LATEX which is the most widespread macro package and should be sufficient for most mathematical notation.

4.2 Control-Sequences and Mathematics

As said in the previous section, one gives instructions to TEX on how to typeset a given document by using control sequences. Control sequences starts with \ and consists of words that are more or less descriptive. Some control sequences take arguments enclosed in { and }. These symbols are also used for scoping and grouping which will be treated more thoroughly in the next section. The rest of this chapter presents some of the commands used when typesetting mathematics and describe the model used for expressions.

4.2.1 Different Modes for Typesetting

TEX uses different modes for typesetting a document. In each mode, the typed letters have different meanings and the rules for style and spacing differs widely. Text mode is used for typesetting ordinary text and is the default mode of operation. In addition there are two modes for mathematics—inlined and displayed. Their meaning is as defined in chapter 2. The inline mode is used by entering the formula between $ signs and the displayed mode is obtained by entering mathematical expressions between two dollar signs ($$). It is up to the author to choose the most appropriate mode for displaying the mathematics in the document.

TEX handles much of the layout decisions itself, just like a renderer for MathML but still, the system is very tied to typesetting for printed media. It automatically follows the convention for spacing and placement, such as moving limits differently depending on the chosen mode, and adjust spacing to reflect how tight different operators bound. Other constructs where the author typing in MathML needs to use external style sheet languages are to some extent built into TEX. It automatically numbers equations and provides registers and counters that can be used in macro definitions to create cross-reference tables of definitions, theorems, proofs, etc.

4.2.2 Operators and Operands

Operands and operators can be typed directly as characters on the keyboard if they are available on the user's keyboard or in the used character set. If not, every operator and operand character have a corresponding control sequence as well. In plain TEX there is about 900 different control sequences. The available characters depends on the available fonts. In principle, one can draw images corresponding to the glyphs one wants to use, and assign control sequences or characters present on the keyboard to them and type the manuscript as usual. As long as the metrics of the font is known to TEX, the document will be typeset fine. This is due to a very flexible design where characters belong to different categories and the correspondence between glyphs and characters can be controlled by the user.

The elementary functions like $\cos x$, $\sinh x$, $\log x$ etc, are also represented by control sequences in TEX. They could be written using normal text but the control sequences exists for convenience when in math mode since words typed directly will come out as a product of symbols in italics, e.g., $cosx$. They are not stretchy so if the argument is a compound expression, parenthesis should be used as in $\cos(\alpha + \beta)$. It would not be correct to use grouping symbols as delimiters as is the case with for example radicals and fractions.

Fencing symbols are stretched automatically to the size of the contained expression. Almost any parenthesis-like symbol can be stretched and one precedes the symbol with the control sequence \left to the left of the expression and \right to the right of the expression. These control sequences must occur in pairs to allow TEX to determine what expression is included but it is possible to use any delimiter as argument or a non-delimiter like a period to denote an invisible fence.

4.2.3 Spacing, Font Styles and Sizes

TEX has many different control sequences for styles but not all symbols are available in each style in all fonts. Available styles are typewriter/mono-space, bold, italics, smallcaps, blackboard, fraktur, script, calligraphic, sans serif, slanted, upright/Roman, bold-italics and variants in math mode and text mode of bold and italics. The standard font in TEX is called Computer Modern Roman. Packages exists for other fonts as well like Times New Roman, Palatino, etc.

Sizes can be absolute or relative. This report is typeset in 11 points. It is possible to specify sizes in other metrics as well, such as pica, inches, centimetres and so on. Size names can also be logical like footnote size, script size, scriptscript size for nested subscripts, etc. LATEX has logical sizes ranging from small, regular, large, Large, Huge, HUGE etc. Fonts can be scaled but a font in five points scaled to double size, like `this text` doesn't look the same as a ten point font as in this text. The reason is that every font has a design size and scaling should not deviate too much from the design size. Most often design sizes are available in geometrical steps of 1.2 or $\sqrt{1.2}$.

Spaces in ordinary text can be controlled as well, both absolute with the same measures as sizes, and relative using control sequences for thin space, medium space and thick space. There is also a negative thin space. By using these control sequences after each other, spacing can be fine-tuned. The automatic spacing in TEX works by partitioning a formula into atoms. A formula is a list of atoms of eight different types: ordinary, large operator, binary operator, relational operator, opening delimiter, closing delimiter, punctuation and inner, for a delimited subformula. For each combination of atoms, there is a table lookup to determine the spacing between them. There are different tables for scriptscript style, script style, inline style and display style. To handle line breaking similar tables assign weights to penalties depending on how bad or good it is to break a line after or between atoms of the different type. For example, it is better to break after a binary operator than a relational operator using the terminology from the spacing algorithm.

4.2.4 Grouping and Scope

TEX uses grouping for two different purposes: One is for setting scope to limit the effect of certain control sequences like changing typeface. Another is to construct a group or box to express subformulas for higher typesetting quality, or to tell TEX how much an accent should be stretched. In all cases, the grouping characters used are the { for starting a group and } for ending a group. The visual model used internally by the typesetter is based on boxes that can be raised, lowered or just placed next to each other with different spacing surrounding them. The width and height of a box is used for calculating the length of a fraction bar, what size of delimiters to choose or how much an operator shall be stretched in different directions. To construct such boxes, the grouping characters are used. The following sections will show different use of grouping in mathematical formulas. When doing calculations one can often view the groups as expressions inside parenthesis but not always.

4.2.5 Scripts and Limits

Superscripts are written using a circumflex (^) and subscripts are written using an underscore (_). They belong to the previous character so the base expression are not normally indicated. If the author wants to indicate it, the expression shall be enclosed in a group using { and }. Every symbol can have exactly one subscript and one superscript and they can be written in any order. Associativity is not used so if they are nested grouping must be used to indicate the correct level. Grouping must also be used if the script is more than a single character or digit (intrinsic control sequence). There is no such notion as atomic subexpression or implicit termination indicators. This leads to a very strict notation at the price of extensive use of grouping.

The subscript and superscript notations are used for limits as well when they appear directly below or above symbols. For some operators, such as integrals, limes, summations, etc, one can choose using special control sequences if limits or postscripts shall be used. TeX also makes intelligent decisions automatically what gives the best typography. For example, in inline typesetting one usually prefers scripts to the right of an expression whereas in display mode limits might be more appropriate to "move them away" a bit when integrands or summands are very complex. Prescripts are constructed as postscripts to an empty group immediately preceding the intended base expression. Staggered multiscripts can be achieved by combining empty groups/boxes and carefully choosing the order of the scripts.

4.2.6 Fractions and Radicals

Fractions and other constructs with a similar layout in TeX and LaTeX can be expressed in two different ways. The first uses the control sequence \frac which takes two arguments, one numerator and one denominator. Each argument is a group whose content determines what should be written above and below the fraction bar. The groups can contain fractions themselves to construct compound fractions. How long each fraction bar shall be drawn is determined by the widths of the boxes with the content in the groups. The choice between horizontal or bevelled fraction bars is made by TeX depending on the complexity of the fractions involved.

The other construct uses the \over control sequence. This sequence does not have any arguments. Instead, the entire fraction is wrapped inside a group and the \over sequence is placed somewhere within the group. Everything inside the group and to the left of the command belongs to the numerator and everything to the right goes to the denominator. Instead of viewing the groups as arguments to a fraction bar, the group can be seen as a fraction delimiter. The former notion is a prefix form of the fraction operator and the latter is an infix form of the fraction operator. This command also let the writer control the thickness of the fraction bar and there also exists similar commands for writing binomial coefficients, Legendre symbols and so on. One can control what kind of delimiters will be placed around the fraction-like expression. These delimiters should not be confused with the grouping symbols that occurs around the fraction and is a kind of delimiter as well. The control sequence \atop is used for fraction-like constructs without a bar and \overwithdelims is the most general for placing delimiters around the content. Vertical piles of symbols or expressions are called stacks in many mathematical computer languages.

Square roots and other radicals are written using the control sequence \sqrt. The kind of radical is indicated by placing an optional argument following the root symbol and the radicand is placed in a group, or box, immediately after the symbol. The width and height of the root symbol depends on the contained subformulae and the root symbol exists in different sizes to choose among. The choice is automatic and expressions involving radicals can be nested.

Chapter 5

Other Formats for Representation of Mathematics

This chapter describes some other commonly used formats for describing mathematics and other highly technical documents. The first section examines some of the major softwares and standards for representing formulas. The next section talks about common Braille codes for mathematics. Many tools and Braille systems which is very much used today could not be included due to the difficulty of finding detailed information. This means that Mathematica, Derive, Scientific Notebook and other softwares are not covered. This also applies to the German Braille standard for mathematics called Marburg Braille and other systems like the Spanish and French ones.

5.1 Electronic Formats

5.1.1 OpenMath

OpenMath is a standard for representing mathematics and originated from the computer algebra framework. The main motivation was to develop a standard for exchanging mathematical computations between such tools. This means that the major goal is to represent the semantics of mathematics, not presentation. The standard is currently in version 2.0 and work on it begun in 1998 in Finland at the university of Helsinki. The main difference from MathML is that there are no presentation elements and the scheme for representing the meaning of mathematical symbols is closely related to, but much more elaborate than, the content elements present in MathML. OpenMath slowly finds its way into the area of mathematical typesetting and information databases as well. An advantage with a semantic representation is in information retrieval systems where you can find the formula $\sin(2x) = 2\sin x \cos x$ even if the variable used in the search query was y rather than x. Using OpenMath for Braille production would make it possible to change the used variables according to the wishes of the reader.

OpenMath bases its representation on something called OpenMath objects. There are objects denoting integers, floating-point numbers, strings, symbols, function application, etc. An OpenMath-aware application is called a phrase book and is able to transform its mathematical objects between the internal representation used privately and a stream of bytes representing OpenMath objects in any of the encodings defined by the standard. The standard defines three encodings: An XML encoding, a functional encoding similar to Lisp, and a binary encoding meant for machine-processing and efficient transfer over networks.

Since OpenMath must be strict in expressing the semantics, one cannot just rely on the fact that $\log_a b$ means the logarithm of b using base a just because the written symbols usually mean it. Instead, each symbol, not denoting a variable, is defined by reference to a content dictionary. A content dictionary can be an arbitrary collection of symbols but the OpenMath committee defines a large number of content dictionaries for a broad range of topics ranging from logic and group theory to calculus and simple high school mathematics. A content dictionary entry for a symbol contains both a textual description, possibly including a reference to a standard work in mathematics, together with a formal definition like $log(a, b) = c \to a^c = b$.

5.1.2 MathType

MathType is a program made by Design science and is the successor to Equation Editor. The main use of Equation Editor was as an extension to Microsoft Word for producing mathematics in word-processor documents. MathType has evolved into a mathematical framework where users can define translators in a language called TDL, translator definition Language. This is an open format and there exists a large number of translators between MTBF, the binary file format used in MathType, and to target languages like for example TEX and MathML. Design Science is also the company behind Math Player, a plug-in to Microsoft Internet Explorer for viewing MathML on the web. The translator language first appeared in MathType version 4. Earlier versions used a hard-coded TEX translator for copying or pasting equations to and from the clipboard. To make editing of MathType equations possible even after translation to and from other formats, they are preserved in their original form as comments inside the target language. The binary code for the original equations are stored using a variant of BASE-64 encoding.

The equation structure used in MathType is based on templates partitioned into slots. Templates can also be given parameters separated by slash. The structure is hierarchical and the top-most element is an equation element that serves as a root container. There are templates for roots, fractions, scripts, etc. Embellishments, such as fences, overbars and hats, are treated as templates as well. Script and limit templates exist in a scanning and non-scanning version. The non-scanning version is used for translation into languages where the mark-up of scripts only contains the script itself which is suitable for some target formats. the scanning versions scans in both directions to determine what expression is affected by the script or limit. The nearest item is either a slot or embellished, fenced, expression. The scanning version suits other target formats where the base expression is treated as arguments to the script and limit operators or functions. slots can be grouped by a pile operator to form a vertical stack. There are 35 different variants based on the pile's position in the equation hierarchy and the alignment of slots and the number of slots in a pile. It is possible to output code before, between each pile and after all translations of the piles.

The slots form a horizontal group of characters. The slot is very much related to grouping constructs like boxes in TEX and the `mrow` element in MathML. It is possible to define two versions of each equation translator, one used for inline style and one for display style. Likewise, there are 16 different slot translation commands depending on size in the original equation, such as script or limit, the position in the structure hierarchy, and the contents of the slot, for example if it is empty, contains a single character with or without embellishment, and so on. The slots are numbered from 1 and up inside a template so that a translation rule can refer to them. Translation rules are applied recursively and with different priorities and unmatched parts of equations are left untranslated. Inside slots are characters. They can be translated differently depending on font size, font style, or if it is in math mode or text mode.

Translation rules can be applied to entire runs of characters of similar properties as well. Runs of characters can be sequences of adjacent characters in a special font, but also characters that shall be treated as a single unit such as numbers and functions. In general, runs can be named and used to translate individual characters differently depending on what run they belong to. Runs may also be surrounded by prefix and postfix codes. There are 12 logical styles in MathType, for variables, functions, symbols, vectors, numbers and text, Greek upper- and lowercase letters, user-defined, etc. One can also address font names such as Times New Roman. There are four typefaces, normal, bold, italic and bold-italic. Point sizes can be absolute or one of the seven logical sizes such as full size, subscript, subsubscript, subscript and normal symbols, 2 user-defined, and so on. A run can also be defined in the purpose of being a single entity, for example a superscript, as a way of grouping.

Matrices are translated by defining rules for the whole matrix, for a row, and for an individual element. There is also a version that affects only the last element in the row. If the matrix has partition lines one can define rules for dashed, solid and dotted lines in vertical and horizontal direction respectively. Code serving as prefix and postfix to elements, rows and the entire matrix can be output as well. different versions for different alignments of elements exist as well in a similar manner as slots in piles.

5.1.3 ISO 12083

ISO 12083 is the successor of the AAP/EPSIG standard, developed by Association of American Publishers and Electronic Publishing Special Interest Group. ISO 12083 is a standard adopted by both ISO and NISO in the 90ies. It is SGML-based and is constructed as an archiving format for electronic documents. It consists of four DTDs for books, articles, tables and math formulas. The mathematics representation is purely presentational and not semantic in its nature like OpenMath but as with MathML, the connection between mathematics notation and meaning is strong and one can argue that it is possible to do some semantic reasoning.

The DTD contains all common styles: bold, italic, sans serif, typewriter style, smallcaps and Oman. They are elements instead of attributes. The supported alphabets are Latin, Greek, Cyrillic, Hebrew and Kanji. The top elements for formulas are called `formula` and `dformula` for inline and display style mark-up. There is also a `dformgrp` element for groups of display-style formulas that can be aligned or numbered for later reference. There exists about 20 elements for encoding of different mathematical constructs. Most are layout-oriented controlling alignments, variant of symbols, fencing and embellishments but there are also more semantic ones like `fraction` with its `num` and `den` elements and the `radical` for marking up radicals. The alignment, thickness and type of fraction bar can be controlled using certain attributes. The `radix` and `radicand` elements control the kind of root and the radicand expression.

There are also an `array` element for array-like structures and attributes for setting row and column alignment and separators. Spacing, horizontal and vertical, and line breaking can be marked up as well. Spacing is set in millimetres. Subscripts can be called inferiors and superscripts can be called superiors and that is the terminology used in ISO 12083. They are marked up with the `inf` and `sup` elements respectively. Scripts are positioned relative to the base expression by using the `location` attributes with values `pre` and `post` together with the `arrange` attribute that can be set to either `compact` or `stagger`. Embellishments are controlled with `top`, `bottom` and `middle` elements and the alignment is set with the `align` attribute.

Fencing is marked up using the `fence` element and the kind of fence symbol to the left and right is set by the `lpost` and `rpost` attributes. The `style` attribute controls if the symbol shall be single or double. There is also a system using marks and references with IDs to control size relative to other elements to express stretching. Different kinds of overlines and underlines are marked up using the `overline` and `underline` elements and there is a `box` element for arbitrary box drawing with different dashes around expressions or for drawing different arrows between expressions. Groups are called subformulas and grouping is done with the `subform` element.

5.1.4 Unicode Nearly Plain-Text Encoding of Mathematics

Unicode is a standrad that tries to achieve a character encoding where all used symbols today shall have a unique and ambiguous representation in electronic texts. Today Unicode covers about 100,000 characters and the next version to be released is 5.1. An attempt is also made to include mathematical symbols though it is sometimes hard to tell if two different glyphs are just pure stylistic variants of the same symbol or if the different glyphs really indicate different semantics. A compromise is used where not too many symbols shall occur multiple times but it should be possible to keep semantics as far as possible by just looking at the assigned character code. Unicode 3.0 first introduced about 300 different math characters. In 3.1 about 1000 more was added and now there is about 2000 of them. The fact that semantics shall be reconstructable in very different typographical contexts gives guidance in designing a Braille system where symbols also will be present with a different layout. The reader can find more information in Unicode Technical Report 25. For accessibility and automatic evaluation of mathematics, the invisible operators for multiplication, addition, comma separator and function application exists as well.

Unicode Technical Note 28 describes an attempt to encode mathematics in a way which is as nearly plain text as possible. The notion uses parenthesis and other grouping symbols to achieve an unambiguous one-dimensional representation of otherwise two-dimensional constructs. This is called the linear format. For example, fractions are surrounded by parenthesis and the different fraction bars are written infix as is often done when convertin fractions to a notation using the solidus (/). The scope of subscripts and superscripts range as long as there are non-operator characters or a space occurs. There are also special symbols for certain common digits in subscript or superscript position, like was the case in Swedish earlier Braille system. Such symbols for simple fractions exists in the ASCII code as well. If expressions are complex involving operators, grouping is used. This format is called The linear format to distinguish it from The built-up format that software can construct from the linear format, presenting a more graphical view. Rules are constructed to transform between the formats. For example, a fraction can be presented as numerator and denominator by removing the outer parenthesis if any and splitting at the fraction bar symbol.

In general, the format makes heavy use of associativity and precedence rules together with some more principles about well-formed expressions to avoid grouping as much as possible as we examined in chapter 2. For example, if a fraction contains only operators and operands with higher precedence than division, no grouping is needed. Should the author wish to have parenthesis remain in the built-up presentation, just double the outer ones. This is in fact a correct mathematics notation in the linear format as well, preserving semantics.

5.2 Braille Systems for Mathematics

5.2.1 Nemeth Braille Code

The Nemeth code is one of the most used Braille codes for mathematics and transcription of other technical material in United States. It was invented by Dr. Abraham Nemeth who started working on it in 1946 and the first version was published 1952. It has than undergone some small revisions 1972 and 1998 when Braille codes for writing advanced chemistry was added. The main purpose for inventing the new writing system was due to shortcomings in the earlier used Taylor math Braille code. One of the most severe restrictions was that it was not possible to write arbitrary exponents. While there were special signs for the superscripts in x^2, x^3 and x^4, if you needed to write x^5 you were unlucky. Another criticism was that some people thought the use of grouping signs was to extensive. Fractions were written in a notation more like programming languages where grouping signs were used around numerator and denominator. During the last years, Nemeth has been updated and the name is changed to NUBS, Nemeth Uniform Braille System. There is also an accompanying method for spoken mathematics that is suitable for reading complex expressions aloud.

One of the major differences between Nemeth code and literal Braille is that digits are written in a lowered form using the lower four dots of the cell when in Math context (see below). This makes it convenient when letters and digits are mixed like in algebra. However, punctuation symbols also occupies the same dots so instead they need a punctuation indicator when the meaning of those symbol changes. This also implies that operators like $+$ and $-$ need prefixes, dot 5 in Nemeth code. In fact there is a numeric indicator as well which is used when digits appears directly after a whitespace mainly for readability, and in literal context where digits are not written using the lower part of the cell. The fact that certain letters in mathematics are printed in italics is not indicated but sometimes certain letters need to be prefixed with a special symbol to indicate that it is a literal letter. This is most useful when abbreviations, acronyms, or the short form or contraction of a word is used in the mathematics context. There exists symbols for expressing certain typefaces as well, namely bold, italics, sans serif and script. They should only be used when they convey mathematical meaning such as vectors in boldface or sets in scripts.

Nemeth is not only a code for mathematics, it also prescribes how to handle literal Braille in printed material of scientific nature. There are three different modes, or contexts, that affect the interpretation of the Braille codes. There is a mathematical mode, a literal mode and a hybrid mode. Thus, the number 20 is printed differently if it appears in a literal context than in a mathematical equation as explained above. The hybrid form is used when letters and digits are mixed such as in the 20th of Sept. At first, the switch between the math and literal modes were indicated using double space characters. In 1965, the toggle prefixes to change between math and literal context was changed to prefixes involving the letters l and m for greater readability. It is possible to represent characters from Greek, Russian, Hebrew and Latin alphabets as well. Accents and bars can be placed on arbitrary symbols and are called modifiers. Modifiers are not restricted to bars, they can be entire expressions if necessary.

In the Nemeth code, fraction delimiter signs, marking the start and end of a fraction is used instead of general grouping signs surrounding numerator and denominator. If fractions are nested, the innermost fraction is surrounded by one delimiter, the next outer one is two delimiters and so forth. This way, the reader knows the coming complexity. Even a simple fraction like $\frac{a}{b}$ requires delimiters. Mixed numbers like $1\frac{3}{4}$ should not be viewed as a fraction. Nemeth divides fractions in three categories: Simple fractions that are not nested, complex fractions with a nesting depth of 2 and hypercomplex fractions with higher nesting depths. The slanted and horizontal bars have different representation.

Subscripts and superscripts are indicated by writing special signs that precedes the subscript or superscript. The end is marked by a dot combination that resets the position to the baseline. Nested sub- or superscripts are indicated by repeating the start code to inform the reader of the complexity but rather than repeating the outermost code, it is the opposite. This means that it is rather the nesting level or scriptsize that is indicated. There is no braille code to restore the position when one subscript or superscript ends. Rather, each operand or operator is indicated by repeating start codes as needed to show position relative to the baseline. The Nemeth code also distinguishes underscripts and overscripts from subscripts and superscripts and a symbol to denote end of under- and overscript exists. Limits on summations and integrals are also counted as scripts but not scripts in chemical formulas.

Radicals are treated in much the same way as fractions. the order of the root, such as a cube root, is written with the digit 3 as superscript to the radical sign. Elementary functions, like log and cos are given special codes to avoid misunderstanding if the same combination of letters occur in a product since italics were not shown in these cases.

Nemeth code is considered hard to parse with computer programs. The reason is that the language is context-sensitive instead of context free. This is because nested superscripts are indicated differently depending on the nesting level. Also, the system is not what is called in compiler theory LALR(k). This means that there is no upper limit on the number of symbols that needs to be examined, the size of the look-ahead, to gain full information about context. This is a large draw-back nowadays when Braille production should be automated. However, an approach using denotational semantics and logic programming has proven to be useful.

5.2.2 LAMBDA Braille Code

Lambda stands for linear access to mathematics for Braille device and Audio-synthesis. It is a research project financed by the European union and started in 2002 and was scheduled to end in 2006. The goal is to develop a linear textual sequential notation for mathematics suitable for rendering in linear media like Braille and audio. Tools were developed to enable editing and easy conversion to and from common formats and print in Braille and normal print. The notation is based on MathML.

5.2.3 Marburg Braille Code

Marburg Braille specification is a European de facto standard that prescribes everything from the dimensions of the dots and spacing in the cell to the representation of the different symbols like letters, numbers and punctuation symbols. Marburg Braille is used on pharmaceutical packages. The origin of Marburg Braille is German and like Nemeth its math Braille code is context sensitive.

5.2.4 Unified English Braille

The many different writing systems, or Braille codes, used throughout the world makes it difficult to read books produced in other countries even if the language in the book is well-known for the reader. This is especially true in English-speaking countries. In US, there were three different

Braille systems in use in 1990. The project to construct a unified Braille code was born in 1992. The project switched name to Unified English Braille to specially indicate the fact that English Braille was the considered one. After almost 13 years of research and debate, the International council of English Braille gave a go-ahead for the unification of English braille codes. They considered the standard to be sufficiently complete and mature to be recognised as a standard code for English Braille. In October 2007, only four of the seven IECB countries have replaced their earlier system, Australia, New Zealand, South africa and Nigeria. One major objection to UEB is that it does not handle mathematics and computer science as compact as would be desired for a comprehencing reading of such literature.

The writing of digits using exclusively lower parts of the cells is skipped. This gives rise to all the necessary indicators for distinguishing digits and letters a to j but the difference from literal Braille is eliminated. Overall, the people behind UEB devoted quite a lot effort to make sure that it should be totally unambiguous where a symbol ends and another next starts. This is done by relying on a prefix-root principle. The familiar number sign, and dots 4 to 6 are reserved for prefixes. All others are root symbols. Two adjacent root symbols cannot belong to the same printed sign and a prefix must always be followed by a space or a root. Some exceptions exists as well but they are few. This also gives the possibility to assign special meaning to multiple occurrences of prefixes or a prefix followed by a space to denote a "poetic line-break" where the printed original breaks the line but not the Braille copy for space-saving reasons. Such line-breaking characters should not be used when there is an array arrangement in mathematics where the two-dimensional structure makes it clearer and easier to read if kept. Stretchy fences are indicated with a prefix attached to the original symbol and repeated on every line in the Braille representation to keep the purpose from the printed original.

Typefaces, called typeforms in the UEB reading rules, can also be indicated if they affect the meaning of the text. There are symbols for five transcriber-defined styles that can be used if the standard bold, italics, script or underline is not sufficient. One use could be where colours are used to indicate various parts of speech. All indicators for typeforms, as well as for capital letters exists in different variants extending over a single symbol, a word or symbol group, and a passage which is an arbitrary block of text requiring an explicit terminator. UEB also defines a special line-drawing mode that can be used in for example structural formulae in chemistry or long division in arithmetic. Symbols for solid, dashed and dotted line segments, both horizontal, vertical and diagonal, exists as well as corners and crossings of various kinds. Reserved symbols that can be defined by the transcriber exists and prefixes to introduce transcriber's notes where they can be described.

Accents and diacritical marks are written before the symbol they apply to (prefix) to indicate a different pronunciation when reading aloud, in contrast with for example Nemeth where they always are written after (postfix) like subscripts and superscripts to the right. There is also a ligature binding symbol written between the symbols that are tied together. They are used when the ligatures have meaning apart from pure visual typographic effects. Accents like u with umlaut should not be confused with mathematical modifiers like the second derivative of u. Mathematical modifiers can be applied to expressions, not just single symbols. If some text is written in a Braille system other than UEB, there are delimiters to express this fact as well as what Braille system is used. This could be the case when music notation is mixed with standard text, or when excerpts from foreign alphabets are written in their native Braille code instead of being transcribed to English letters.

Subscripts and superscripts are written before or after the base expression depending on the appearance in print. Special symbols are used for indices directly above or below. The scope is the nearest item and an item can either be indicated by using a grouping indicator or be implicit according to different rules. Worth noting is that if the expression should require a delimiter for other reasons such as being a fraction, radical or a parenthesised expression, the delimiter for the index can be omitted. Situations where the delimiters are needed is for example if a half-open interval occurs as integrand or two parenthesised expressions occur as index separated with a comma. Subscripts and superscripts can be nested and the indicators for the baseline shift are relative to the previous position. This is like the English and Spanish Braille math code but different from Nemeth's absolute indicators.

Fractions are represented using open and closing fraction delimiters together with a symbol for a horizontal fraction bar. If the fraction is simple, with only a number in numerator and denominator, the delimiters are left out. With this scheme of writing fractions they can be nested in a straight-forward way. Slanted fraction bars are displayed using a slash and since parenthesis are needed in print as well, no extra care must be taken. Delimiters used in the printed original are simply reused in the Braille copy. Radicals are written using open and closing delimiters in all cases. There are two different radical symbols, one without vinculum and one with vinculum where it is stretched over the radicand. This means that nesting is straight-forward. The same definition for item used in conjunction with fractions is used when placing arrows or bars above or below expressions. No precedence rules exists so in unusual cases where one needs to distinguish $x^{\overline{y}}$ from $\overline{x^y}$, i.e. if the bar applies to y or x^y, grouping symbols must be used.

UEB contains an elaborate method for construction of shapes and arrows. There are symbols for bold and normal arrows and eight terminators depending on the direction of the arrow. Looping arrows, arrows that cross itself or form enclosures, are treated as shapes. Different symbols for the length of the arrows are also defined. Shapes can be unfilled, filled or shaded and it is indicated for transcriber-defined ones as well as for the ones that have their own Braille codes defined in UEB. All regular polygons can be expressed as well as other common ones. If an assigned code exists for a shape, that code should be used in favour of transcriber-defined ones. shapes can also be composed of other shapes using certain prefix indicators and terminators. Combination symbols for superposition, horizontal and vertical juxtaposition and physical enclosure are constructed. Each of this signals a combination of the previous and following items (infix use) and items are defined as earlier. There is also a special series of symbols that are left as to be defined by the transcriber. This avoids defining Braille codes for symbols that are very infrequent but used extensively in some texts. Despite this fact, there are many symbols defined from chemistry, physics and computer science in addition to the mathematical constructs.

5.2.5 The French Braille Code for Mathematics

Chapter 6

Conclusion

6.1 Braille Representation of Mathematics

The most elaborate question to answer when designing a braille system for mathematics notation according to the goals described in chapter 1 is what should be represented and what may be omitted. The system must be complete and be able to express the logical structure of the underlying mathematics and still be easy to read freeing the reader from thinking of the notation rather than focusing on the presented material. Still, the system must keep much of the concepts and terminology from the layout used in standard mathematics notation. This is to make sure that people reading Braille can communicate with the sighted environment and know how to present mathematical content. In this section, a systematic treatment of the notational concepts introduced in chapter 2 is given with the aim to decide what needs to be included in braille to preserve the mathematical semantics, what has to be left out in practise to keep readability and what mechanisms are available for increasing readability in Braille of the mathematical content.

The Braille system must also meet a very broad spectrum of mathematical content, ranging from simple mathematical constructs in elementary school to very advanced material on graduate level. If both kinds of content shall be covered by the same notation, it is unavoidable that the notation used for elementary mathematics will be more complex than necessary or certain advanced constructs will not be representable at all. A way around this is to define two different systems for mathematics notation, one simple and one advanced. The drawback is that the Braille reader will have to learn a new writing system when studying more advanced material but an advantage is that most people will not need to learn both notations. This approach will not be further developed in this report. It is not easy to decide where the border between the two notations shall lie. Complex expressions involving nested fractions and square roots, for example when solving a quadratic equation or Pythagoras theorem will be encountered quite early in the teaching of mathematics. If a distinct border cannot be drawn, it is necessary that there is a graceful transformation between the systems. This limits the possibility to optimise the two systems for their purpose regarding readability and ambiguity.

6.1.1 Operands and Symbols

Which characters shall have a standardised representation and which ones should only rely on descriptions?

6.1.2 Operators, Fences, Separators and Accents

In natural language processing, accents together with the character it is applied to is often treated as a special character with its unique character code. Some coding systems, e.g., Unicode, also has a single stand-alone accent treated as a combination character. In mathematics, these diacrits are more flexible. Most notably, they can be stretched over multiple symbols. This is the main reason why almost all writing systems for mathematics encountered in this report treat accents as operators. They can be collected in the same category as parenthesis that also can be stretched to enclose expressions. It is clear that such a treatment in Braille will also be the most flexible solution. Construction rules for all combinations of characters and accents will be consequent. In literal text it might make sense to allow for both the method where accents can be applied as an operator and a simplified way of writing that is easier to read where combinations of characters and accents that occur frequently have special Braille codes, like é, ú, å, ä and ö.

6.1.3 Grouping and Scope

Grouping, in the role of determining scope of commands or functions, is a well-known concept in programming languages. An example is when parenthesis are used to enclose arguments to a function, or when brackets or braces are used to give limited scope to variables. When writing mathematics in a linear way using mark-up or a functional notation, grouping is often implicit. In programming languages, parenthesis of different kinds are commonly used to express grouping and scope. In MathML, the start and end tags serve the purpose of grouping delimiters. In LaTeXthe box model is an example of grouping using braces to tell what operators act upon. Limiting scope of certain control sequences, such as style changes, is another case.

The grouping characters are probably the most important in Braille when typesetting mathematics since Braille is a linear mono-spaced media. It is for sure the most common way to indicate scope of stretchy operators, or other grouping such as fractions, scripts, style changes and so forth. The grouping Braille codes can be designed in many ways. One alternative is to use different grouping symbols in different contexts, such as one for fractions, another for radicals, a third for scripts, etc. This increases the number of required Braille code in the mathematics representation but the reader does not need to remember the correct nesting depth and order of the sequences indicating the purpose of the grouping.

Another alternative, which could be extra meaningful when it comes to indicating changes in typefaces is to let the starting and ending grouping symbols to indicate the purpose of the grouping instead of a code sequence before the start grouping symbol. This makes it possible to express nested typefaces in an optimal way, not requiring the termination of more typeface changes than necessary. A combined approach—where the grouping symbols always are the same but the ending symbol is always followed with a Braille code indicating what ended is maybe both easy to read and flexible.

6.1.4 Spacing styles and Sizes

Spacing is mostly used to increase readability. The most common example is when operators with higher precedence have less spaces between them and the arguments to remind the reader of the tighter binding and higher priority. This can be used in Braille as well. A space in Braille always has the same size so other usage to communicate structure is maybe not possible. Some small spaces in the printed material, such as the space between the integrand and the integrator should probably be left out in Braille.

Font sizes is also important for readability and its use serves different purposes. One example is the nesting of functions within functions where the readability increases if the sizes are varied small amounts for the different nesting levels of the parentheses. This applies for the square root sign as well. Stretchy operators can also be seen as glyphs in different sizes and is another example of where sizes are varied for readability but more importantly, to communicate grouping. This grouping is semantic as well, i.e. the meaning of the content will change if the size of the operators were reduced to a single size. However, not all font size variations are significant for the understanding—the two versions of the integral sign used in displayed typesetting and inline typesetting always have the same meaning.

One solution would be to have a fixed set of sizes available for such symbols and assign different Braille codes to each of them. This reduces the possibilities to keep the Braille system simple. It also means that certain expressions will not be possible to write in braille in a consistent way but that is comparable to all typesetting with font limitations. A further approach may be to use the graphical capabilities of modern Braille printers to construct arbitrary stretched operators. For online reading using refreshable Braille displays this would not be that easy yet. If grouping is what should be communicated, one can either use spaces such as line breaks around, and/or indentation of, the content inside a stretchy bracket, or in the case of radicals or wide accents, combine them with nested grouping characters used in a similar way as parenthesis. Both approaches means that it is enough with a single representation for a given bracket. To avoid the use of grouping one can use the method in UEB by defining items that forms implicit groups or the Nemeth approach where grouping signs and nesting depth are not symbols in its own right but rather expressed by repeating the symbol that is stretched or nested.

Variations in typefaces increase readability in similar ways as the stretchy operators. They both convey meaning of an expression as well as making the material more easy to perceive visually. The difference between a product of two single-letter variables, and a variable consisting of two letters is mostly communicated with the aid of typeface conventions. Specially when space cannot be used in a fine granularity. This is also the case with elementary functions whose name involves multiple letters. However, always showing changes in typefaces in a Braille material would make the material hard to read. When reading with eyes, small variations of spacing and sizes will help the reader but that does not mean that indicating all such variations will be beneficial when reading by touch.

Fortunately, mathematicians do not tend to use both x, y and xy in the same text. Ambiguity may still arise though so in the end it is either the Braille producer, or the producer of the semantic mark-up, that must know the subject sufficiently well to be able to take appropriate actions when needed. There are essentially two main approaches: either one adopts certain conventions like never show that single-letter symbols are written in Italics and take appropriate actions when ambiguity occurs—or one uses grouping characters together with Braille codes to indicate the start of text in a new typeface. In the first case, one can add an extra foreword to the Braille material summarising the conventions used.

6.1.5 Scripts and Limits

There are a lot of different models for scripts and limits examined in this report. In MathML the base expression is marked-up and to each base expression there exists exactly one superscript and one subscript. In TEX, the script is associated with the closest preceding symbol and their can be at most one subscript and superscript for it. If multiple levels of sub- or superscripts occur they have to be nested. Also, if the script contains more than one character a grouping sign must be used to indicate scope of the exponent or subscript. In UEB and Nemeth there are codes to indicate baseline shifts. There is a code that indicates end of script or end of baseline shift but there are different approaches taken regarding if the end of the baseline shift means that the position is reset to the baseline or if it is just the effect of the previous shift that is abolished.

With the latter approach, scripts can be nested arbitrarily but for convenience a less complex script should be written before a more complex one for readability. An example is the expression $x_1^{a+b^c}$. With the first approach there must be different codes to indicate the nesting of the script or a code for a new script of the same kind must be treated as not containing an explicit termination code while one can decide if the start of a script of the other kind should mean the end of the previous one or not. Expressions like $a_1^{b_1+c_1^2}$ will be difficult to read with such a convention.

Use of grouping symbols leads to more Braille cells but is consistent to read and understand. If combined with the notion of items in UEB, where grouping symbols can be omitted in certain cases, decent readability is probably obtained. There should also be no need to indicate the start of the base expression in Braille. Scripts to the right of the symbol can be written after the symbol and scripts to the left immediately before the symbol. They will always be followed by a space or located at the beginning of a line. There is no semantic difference between scripts directly above or below a symbol, called limits, or if they are located to the right. It is more of a typographic taste as in

$$\int_0^{1-c} (x^2 - 3x + 4)\,dx$$

or

$$\int\limits_0^{1-c} (x^2 - 3x + 4)\,dx$$

6.1.6 Fractions and Radicals

Radicals can be treated in much the same way as scripts. A braille code can be used to indicate start of a radicand and another code to indicate its end. The end code can be skipped if grouping symbols are used instead. The Nemeth approach with different start codes to indicate nesting level has the same drawbacks as with scripts and limits of being hard to generate automatically.

With fractions we have seen basically two models. One is the MathML way using grouping with a functional approach where there is a fraction sign followed by two groups, one numerator and one denominator. Compound fractions are nested. This representation is difficult to use in Braille since we are used to write fractions in an infix manner and a functional approach will not have any advantages. TEX uses grouping more as fraction delimiters. In each group there can be exactly one fraction bar. Nesting must be used for compound fractions as in MathML.

Nemeth uses the standard way of reflecting the nesting depth in the symbol used for the start of the fraction or the code for the fraction bar. UEB uses nested grouping in a fraction-delimiter like philosophy. It is consistent and together with the notion of items shall be sufficiently readable. Worth to note is that only fractions with horizontal bars must be treated specially in Braille since fractions with slanted bar needs explicit use of parenthesis or relying on an implicit one using some kind of item notation. In elementary mathematics, or when fractions are very complex, a representation that is two-dimensional even in Braille can be used for greater readability. This is not analysed in this report.

6.1.7 Elementary Functions

The elementary functions cause no special action when represented in Braille. In theory, if one chooses not to indicate all changes in typeface it could be possible to miss-interpret a product like cos and the cosine function. This can be avoided by giving each elementary function a special dot combination but it should be clear from the context what the author has in mind. For pedagogical reasons, mathematicians should avoid using three-letter symbols like sin or a product like sin with other meaning than the sine function in a text about trigonometry. It is sufficient to type their names followed by a space and the single-letter argument, or a parenthesis containing a compound argument. In most cases, the space can be avoided. The functions are not stretchy operators and cannot be placed in some special position relative to a fraction bar to indicate implicit grouping. Thus, their standard representation in mathematics is suitable for a linear media without any special treatment at all.

6.1.8 Keeping the Notation Alive

An important aspect, at least for graduate level of mathematics is to be able to construct new symbols in a meaningful way. The dots in a Braille code have very limited possibilities to construct intuitive symbols that mimics the original symbols in the printed book. If the convention of describing symbols as plain text is adopted, this approach can be used where the character occurs very infrequently. In literature with very frequent occurrences of the character, it is important to reserve some Braille codes that can denote arbitrary symbols and let a special foreword in the Braille material explains what these Braille sequences mean in this document.

The problem is not trivial though since one wants to avoid the fact that the same character gets very different descriptions in different books. One approach is to reserve certain symbols that always are user-defined so that no specific notion will be associated with them and to adapt a convention that certain topics in mathematics try to use them consistently. Alternatively, one develops a scheme for describing arrows and other geometrical shapes like in UEB. Using the graphical capabilities of modern Braille embossers is an option here as well but with the same limitations as mentioned earlier for online reading.

6.2 Future Work

This report has made a detailed treatment of the presentational part of MathML. One reason is that it is the most used set of elements to write documents with mathematical content. Another reason is that the DAISY standard base the mathematics representation in DTBook on those elements as well. However, it is clear from the reasoning in this report that it sometimes is necessary to use a notation in Braille that deviates quite a lot from standard conventions in order to keep the content easy to read. This can only be achieved if notation can be adapted to a great extent without losing the semantical meaning of the described mathematics. With this point of view, it could be interesting to look into the semantic part of MathML, called the content elements, and study how such mark-up can influence the Braille notation.

Another topic to look more closely into is what makes mathematics easy or difficult to read in Braille. There are indications that many tactile readers find mathematics difficult. This fact cannot be explained entirely by limited skills in Braille. While one explanation could be that the mathematics notation makes a heavy use of two-dimensional flows and other visual effects, and many teachers teach mathematics in that way, we still need to find ways around this problem. An example is that calculations on paper are done from right to left, people doing calculations in other ways, do it from left to right. Division is another concept that can be managed in many different ways and some of them may suit a linear representation better than others. The most use of such visual descriptions are in elementary school and that time is very important in acquiring a basic understanding and living interest of further studies in mathematics.

The main area to make more research in this case is what makes reading of mathematics hard? One assumption is that the very large number of symbols leads to many different Braille characters to keep in mind. Since the number of unique combinations are few, many characters will be similar to each other. Another assumption is that long sequences of the same character is difficult. If this is the case, one should not use many nested identical grouping symbols like parenthesis but instead use many different grouping symbols in different contexts. On the other hand, this will again lead to many Braille symbols to remember. A third approach is to use more verbal written descriptions of mathematical expressions like what is done in talking books. This might be better if Braille reading in general works well but mathematics is cumbersome and cause trouble due to long lines of “cryptic” symbols.